Software de control en sistemas simulados y con dispositivos reales

Universidad Carlos III – Junio 2012

Autor: Ignacio Garcia Alonso

Directores del proyecto:

Jesús García
Enrique Martí

El autor de este proyecto quiere agradecer el esfuerzo y apoyo dedicado por Carlos Martín Martínez en sus primeras fases. El trabajo conjunto ha sido no solo necesario sino ameno y comprometido. A Jesús García y Enrique Martí por excederse en sus obligaciones como tutores. A mi familia, amigos.

Tabla de figuras

Tabla de contenido

Introducción

El uso de robots teledirigidos o autónomos está proliferando en la actualidad gracias a múltiples utilidades que se han aplicado a la robótica. "Accidentes nucleares, localización de naufragios, exploración de volcanes y viajes espaciales […] Los robots están transformando la forma de vida y trabajo y están expandiendo los límites de la experiencia humana" (Ollero, 2001).

Hoy en día es posible construir un robot autónomo con un presupuesto no muy cuantioso gracias al avance tecnológico y el abaratamiento de componentes. Añadiendo la completa integración con unas soluciones software que, con el paso del tiempo, han perfeccionado sus técnicas, estas soluciones se convierten en un cambio revolucionario en la ejecución de tareas del ser humano.

El proceso de cualquier ingeniero que desee implementar alguna técnica software sobre la robótica, pasa por la extracción, análisis y reacción ante los datos obtenidos de los sensores que el robot posee. Cámaras, sistemas de posicionamiento, sensores laser, acelerómetros… Todas estas tecnologías unidas proporcionan gran cantidad de datos que, debidamente analizados, conllevan a la toma de decisiones consecuentes. Si el objetivo a alcanzar es la construcción de un vehículo no tripulado el grado compromiso con el software de control se dispara.

Es por ello que la complejidad del sistema aumenta proporcionalmente al grado de su autonomía. Los sistemas más sofisticados poseen una solución software que emplea una o varias técnicas de alta complejidad que proporcionan al sistema la autonomía deseada.

El software de control, principal objetivo de este proyecto, es aquel que implementa el ejercicio de respuesta hacia los mecanismos de movimiento dados unos datos específicos. Una buena analogía para ejemplificar este concepto sería considerar al sistema de control como el cerebro, los sensores como los sentidos y los componentes que producen el movimiento (motores en este caso) como las extremidades.

El sistema de control del robot tiene como principal objetivo la toma de decisiones respecto a los datos leídos por los sensores. Una vez tomadas esas decisiones, comunica los motores el siguiente movimiento a tomar. Cuando es ejecutado el movimiento, los sensores vuelven a capturar datos y vuelven a pasar a manos del controlador, este mide el error producido y decide, de nuevo, la siguiente decisión a tomar. Este ciclo se repite continuamente sea cual sea el tipo de robot y las decisiones que este deba tomar.

Este tipo de sistemas ofrecen muchas e interesantes áreas en las que investigar y trabajar.

Este proyecto, así como en gran parte de los casos, el problema que se trata de solucionar es la navegación y movimiento del robot.
Tan importante es el análisis de los datos obtenidos como la reacción consecutiva a ellos. Es este el principal objetivo de este proyecto. El sistema de control que se

desea alcanzar en el futuro deberá tomar la decisión más correcta en el menor tiempo posible dados unos datos. Si se desea conseguir esto, no solo deberá apoyarse en los datos actuales si no en el error acumulado, movimientos anteriores y la previsión de futuros errores.

Para llegar a este objetivo, el siguiente proyecto es un primer paso a la implementación de un controlador PID. Se tomarán datos y se realizarán pruebas para comprobar que medidas son adecuadas para controlar nuestro robot.

Si bien este proyecto no implementa un software de control óptimo para el robot, trata de adecuarse a futuras implementaciones dividiéndolo en componentes y pudiendo ser añadidas o sustituidas a este sistema. Es, por tanto, software de control básico que cumple las expectativas desde un punto de vista teórico y funciona en los casos prácticos sin enfatizar mucho en el rendimiento.

En una fase más avanzada del proyecto, se especializa dicho sistema para el sensor de una cámara web. La labor de este sensor será situar el robot en un plano donde pueda ser detectado un objeto concreto y devolver la diferencia entre la orientación necesaria y la que mantiene el robot. En base a esa diferencia el sistema será capaz de generar un movimiento acorde con los datos obtenidos.

En la actualidad, el desarrollo de un robot autónomo como un vehículo no tripulado terrestre (conocido como UGV) suele implicar la integración de un robot comercial con un sistema basado en código libre. En muchas situaciones, dicho código pertenece a varios colectivos y es labor del desarrollador completar la integración de los diferentes componentes y completar el desarrollo con su aportación al sistema. Por lo tanto, gran parte de este proyecto consiste en analizar, investigar e integrar distintos componentes, plataformas y proyectos al que actualmente se está realizando.

En una última visión global y para asentar la idea principal de este proyecto, la implementación de un software de control en la robótica (tanto para pruebas teóricas como para casos prácticos), la mecánica de trabajo consta de diferentes fases como observación, extracción de datos, análisis, diseño, parametrización y pruebas.

Motivación, objetivos y planteamiento.

El desarrollo de software para robots autómatas es de alto interés para el entorno de investigación ya que sus posibles aplicaciones en el sector comercial son muy amplias. Por ello, proyectos de software de control, a priori, a medio camino entre los componentes físicos del robot y el software final y "útil" del autómata, son de especial interés e importantes para el su correcto desarrollo. Por tanto, la principal motivación de este proyecto es ofrecer un servicio real y útil para facilitar la automatización de movimientos del robot.

Además, deberá cumplir los requisitos como software de control y realizar pruebas reales con sensores físicos en un entorno no simulado.

El objetivo prioritario de este proyecto es ofrecer una base software de control para futuras implementaciones y variaciones que faciliten un correcto desarrollo de unos componentes de software específicos basados en cualquier sensor físico. Además, este proyecto debe implementar uno de esos sensores, en este caso la cámara web.

Como suele ocurrir en estos casos, el proyecto se basa en otros trabajos previos que han sido implementados con la misma intención de continuidad que este. Por lo tanto, una primera fase será el análisis, investigación y estudio de viabilidad de los proyectos anteriores. Es necesario poner bajo observación el software previo de manejo de sensores así como el hardware con el que se va a trabajar. En este caso, para no disminuir el rendimiento y aumentar la latencia del movimiento del robot, se decidió trabajar con los datos obtenidos directamente del sensor y no directamente con el middleware proporcionado por el trabajo previo realizado por (Rebollo, 2010) Tomás Rebollo Balaguer. A pesar de esta decisión, el proyecto realizado por Tomás Rebollo Balaguer es una gran influencia y ayuda para la consecución de objetivos de este proyecto ya que la investigación de los sensores y la plataforma Player/Stage es necesaria tanto para la fase simulada del software de control como la implementación del sistema en dispositivos reales.

Una vez realizada la fase de investigación de proyectos anteriores se elabora un diseño del software de control bajo los requisitos anteriormente mencionados.

Tras su diseño, la implementación del mismo bajo el entorno simulado Player/Stage usando sensores simulados genéricos (como el GPS que da coordenadas relativas). Esta fase conlleva a la experimentación y a la realización de pruebas del sistema implementado y a su consiguiente parametrización para alcanzar el más correcto de los resultados y elaborar unas conclusiones útiles. Esta fase tiene también como objetivo declarar el sistema implementado como útil y decretar si cumple sus objetivos.

Luego, dejando atrás el trabajo simulado, se trabaja con un sensor real (en este caso una cámara web) para incluir en el proyecto una implementación real de un sensor físico. Esta fase del proyecto incluye también su correspondiente apartado de experimentación y pruebas.

Explicación del trabajo cooperativo realizado.

Durante las primeras fases de este proyecto se ha trabajado conjuntamente con Carlos Martín Martínez, cuyo objetivo en su trabajo está necesariamente ligado al de este proyecto. Esto implica que durante la fase de investigación y diseño controlador se trabajó junto a Carlos ya que el objetivo primordial era el mismo.

Fue durante la fase final de diseño e implementación cuando se empezó a bifurcar ambos trabajos para convertirse en proyectos diferentes.

En la figura que se presenta a continuación se puede ver un diagrama de desarrollo del trabajo conjunto.

Los procesos resaltados con color naranja pertenecen al desarrollo cooperativo con Carlos Martín Martínez y los procesos resaltados en azul pertenecen al desarrollo individual del proyecto por el autor de este documento.

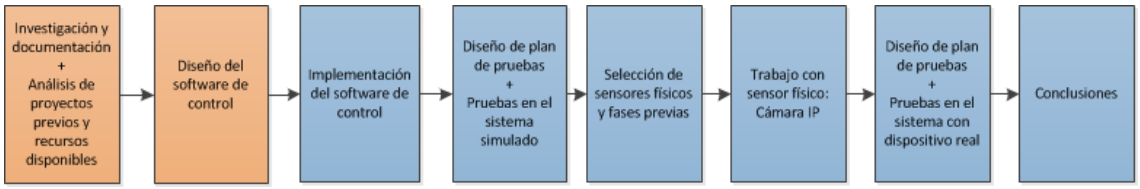


Figura 1 - Diagrama del modelo de trabajo cooperativo

La intención general de la colaboración de ambos proyectos era diseñar un software con la escalabilidad suficiente como para que se pudiesen desarrollar dos componentes de software diferentes para dos dispositivos reales diferentes. En el caso de este proyecto la cámara web y en el paralelo a este el de un dispositivo real diferente.

Por motivos ajenos a este proyecto, el trabajo de Carlos Martín Martínez fue cancelado y, pese a haber planificado su desarrollo de manera conjunta en las primeras fases y paralelamente al final, no podrá contrastarse el trabajo de ambos proyectos.

Fases del proyecto

Durante este apartado se explicará brevemente las fases por las que se pasó durante la realización de este proyecto. Además, se especificará aquellas que fueron realizadas cooperativamente junto a Carlos Martín Martínez.

- Investigación y documentación, análisis de proyectos previos y recursos disponibles: Durante esta fase se dedicó gran parte del tiempo a realizar investigaciones acerca de problema que se debía solucionar mediante este proyecto, además de conocer a fondo las técnicas, teorías y condiciones necesarias para abordar la solución del mismo.

  Las áreas investigadas fueron controladores, vehículos no tripulados, funcionamiento y desarrollo sobre plataformas Player/Stage y estudio exhaustivo de proyectos previos tales como (Rebollo, 2010). Esta fase fue realizada junto a Carlos Martín Martínez y fue una de las más largas de este periodo.

- Diseño e implementación del software de control: Esta fase, pese a ser explicada en este punto como un componente más del proyecto, está desarrollado basándose en una metodología levemente iterativa y, por lo tanto en base a un diseño inicial se implementó prototipos que desencadenaban en nuevos diseños más eficientes y que solucionaban los problemas de los diseños anteriores. Si bien el diseño final y su implementación quizá no sean los más eficientes cumplen con el objetivo principal del proyecto. La fase de diseño fue realizada conjuntamente con Carlos Martín Martínez.
- Diseño del plan de pruebas: Las pruebas diseñadas durante esta fase pertenecen a la parte simulada del proyecto. El diseño de las pruebas debía ser realizado en conjunto ya que se intentaría aportar más diferencias entre ambos proyectos para que el resultado final aportase más información y diferentes conclusiones. La intención durante esta fase era realizar ambos planes de pruebas en conjunto, redactando diferentes baterías de pruebas para, seguidamente, realizarlas por separado y sacar conclusiones paralelas. Debido a la cancelación del proyecto de Carlos Martín Martínez, esta fase se realizó únicamente con este proyecto dando lugar a la experimentación adjuntada en este documento.
- Selección de sensores y pruebas físicas: Durante esta fase, influenciada por los dispositivos disponibles, fue estudiada la viabilidad de los mismos y elegida en consecuencia.
- Trabajo con el sensor físico, cámara IP: Durante esta fase se investigó el funcionamiento de la misma y se implementó los componentes necesarios para la interpretación de datos provenientes del dispositivo. Una vez interpretados, fueron integrados con el sistema ya implementado dando lugar a un sistema simulado con datos de entrada de un dispositivo real.
- Pruebas y conclusiones: Se realizaron pruebas y se sacaron conclusiones propias de cada sistema (el simulado y el implementado con un dispositivo real).

Ventajas y conflictos del trabajo cooperativo

Realizar este proyecto conjuntamente con Carlos Martín Martínez ha sido beneficioso en muchos aspectos. Dos visiones distintas para abordar el mismo problema es, de sobra conocido, uno de los más potentes mecanismos de verificación de cumplimiento de objetivos y aporte de ideas para un proyecto.
Durante la fase de investigación y análisis de otros proyectos el trabajo se definió de manera cooperativa, sin embargo contenía una gran cantidad de trabajo no cooperativo ya que consistía en recopilar, leer, investigar y comprender diferentes documentos teóricos y los casos prácticos realizados por otras personas. Sin embargo, el diseño del software de control fue la etapa más cooperativa de todas. Se diseñó en conjunto y se emprendió especial dedicación en el objetivo principal de diseñar un modelo de software de control que fuese universal para cualquier componente que se quisiese diseñar.

La intención de este trabajo cooperativo no fue otra que aportar dos visiones distintas al mismo problema y, de esta manera, diseñar una solución más robusta e integra.
La mayor parte de los problemas acontecidos durante este proyecto fueron relativos a la coordinación de ambos miembros, así como los conflictos encontrados tras la cancelación del proyecto de Carlos Martín Martínez. Pese a estos inconvenientes, la experiencia colaborativa de este proyecto fue satisfactoria y beneficiosa para su desarrollo.

Metodología de trabajo cooperativo.

Este proyecto, por las magnitudes alcanzables y las posibilidades que ofrece, ha tenido un desarrollo altamente iterativo. Durante la mayor parte de las fases se necesitó rediseñar e incluso redefinir los objetivos el proyecto centrándolos en las necesidades del sistema y viabilidad del mismo. Por ello, incluso durante la redacción de estas líneas, muchos de los componentes del proyecto no tienen un diseño o implementación de carácter definitivo.
Durante el diseño e implementación del proyecto se mantuvo la siguiente metodología de trabajo:

- Análisis e especificación del problema y los objetivos de cada componente.
- Diseño de una solución comprometida con el análisis previo.
- Implementación de un prototipo que cumpla con el diseño y la especificación.
- Realización de pruebas y reflexión sobre sus conclusiones.
- Nueva iteración de la metodología en base a las conclusiones.

Como es de esperar, esta metodología es fácilmente extensible en términos temporales. Por esto, una vez se hubieran cumplido los principales objetivos de este proyecto, se decidió terminar este proceso y continuar con el proyecto.
Respecto a la toma de decisiones y la aplicación de las mismas, fueron todas tomadas bajo consenso obligatorio y total de ambas partes hasta la bifurcación total de ambos proyectos.

Compromiso final de ambas partes.

Una vez realizada las fases comprometidas entre ambos, cada uno de los proyectos toma una línea de trabajo habiendo sido incluso no finalizada por alguno de los miembros. Las modificaciones a posteriori pertinentes en la parte común del sistema, al igual que su mantenimiento, documentación y, en algunos casos, rediseño, corren a cargo del autor de este documento.

Estado del arte

La mayoría de los conceptos presentados en este proyecto son aceptados académicamente y, por lo tanto, no es objetivo final desarrollar nuevas teorías sobre ellos. Este proyecto trata de alcanzar objetivos, facilitar la comprensión y servir de base para nuevas implementaciones haciendo uso de dichos conceptos.

Esta sección es una descripción ligera de los conceptos que ya son aceptados por la comunidad científica y sobre los que se apoya el desarrollo de este trabajo.

Los conceptos a presentar son los vehículos no tripulados, controladores y teoría de control y sensores ópticos.

Durante el trascurso de estas líneas se hará referencia a muchos documentos relacionados con los estudios de estos conceptos. Algunas veces se citará textualmente el contenido de dicho documentos ya que el autor del mismo explica de manera muy precisa algún concepto y ese trabajo no pretende complicar con explicaciones innecesarias un trabajo ya realizado excelentemente por otros autores.

Vehículos no tripulados.

Las necesidades de este proyecto giran en torno al uso de un supuesto robot y un simulador de vehículos terrestres, en este caso, no tripulados. Por definición, un vehículo no tripulado es cualquier equipo mecánico móvil que puede transportar un objeto o sistema con cierto grado de autonomía. Este grado de autonomía viene dado por la carencia de otro sistema externo al robot que ejerza control sobre él mismo. Estos sistemas externos suelen ser personas pero, ante la carencia de estos, los vehículos no tripulados deberán tener la suficiente autonomía para tomar decisiones correctas en situaciones varias.

Dentro de los vehículos no tripulados se puede diferenciar dos tipos: Los vehículos aéreos no tripulados (UAV, Unmanned Aerial Vehicle) y los vehículos terrestres no tripulados (UGV, Unmanned Ground Vehicle). Durante este proyecto se trabajará con vehículos terrestres no tripulados.

Muchos de los avances tecnológicos en vehículos no tripulados provienen del entorno militar por su capacidad de reconocimiento y vigilancia sin necesidad de intervención humana en el pilotaje. Es por eso que muchos de los avances actuales de la industria de la automoción no tripulada provienen de experimentos realizados previamente en entornos militares que más tarde han sido extrapolados al día a día y la vida cotidiana. (SPAIN, 2010) La experimentación e investigación en el ámbito militar proviene desde el año 1960 aunque es actualmente cuando más se utiliza esta tecnología. En el ámbito cotidiano, aún por explorar, se limita en la mayoría de los casos a la ayuda a la conducción en el sector automovilístico (quizá debería considerarse dentro de la rama de vehículos semi-tripulados), ayuda a personas discapacitadas, y pequeños electrodomésticos.

Si bien los avances en la investigación de este campo progresan muy deprisa, el mercado actual absorbe con cuenta gotas las posibilidades que este le ofrece. (Marc Raibert, 2008)

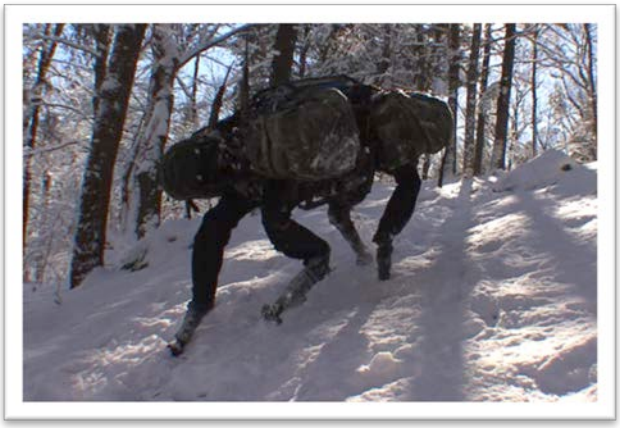


Figura 2 - Vehículo no tripulado Big Dog perteneciente al ámbito de investigación militar.

Uno de los objetivos principales de los vehículos no tripulados es manejar la navegación de los mismos de manera autónoma poniendo especial interés en la orientación y el rumbo de los mismos. Esta ha sido uno de los objetivos principales de este proyecto.

La elección de este tipo de metodología dependerá en gran parte de la necesidad del problema. Para ello es necesario captar datos sobre la situación del robot en cada instante. En base a esos datos, un sistema ha de ser el responsable de la toma de decisiones y decidir el siguiente movimiento. Una vez elegido, un mecanismo ha de realizar dicho movimiento con cierta precisión. Estas tres fases o características necesarias del sistema conllevan a tres necesidades básicas:

- Dispositivos de captación: Estos dispositivos tienen la función de interpretar datos del exterior y enviarlos al sistema. Por lo tanto, existe una conversión del sistema analógico real al sistema digital interpretado. Algunos de estos dispositivos son el GPS, Laser, sensor de movimiento o IMU o la cámara.
- Software de control: Una vez adquiridos los datos, es necesario interpretar es información y, con ella, generar un posible siguiente movimiento. Para ello es necesario un software especializado denominado software de control. Este proyecto centra la su mayor esfuerzo en ese sistema.
- Mecanismo de automoción: Para realizar el movimiento deseado, se han de poner en funcionamiento los mecanismos que provocan el movimiento del vehículo. Dependiendo del tipo de vehículo, estos mecanismos serán motores de ruedas, patas…

Figura 3 - Vehículo no tripulado cotidiano.

Entre los tipos de vehículos terrestres no tripulados encontramos diferencias básicas en los dispositivos de movimientos. Los más populares son los vehículos con dispositivo de ruedas aunque, dependiendo de las necesidades, objetivos y presupuestos, hay muchos estudios acerca de los vehículos de locomoción con piernas (Marc Raibert, 2008).

Los beneficios del los vehículos de ruedas pueden ser la fácil implementación de sistemas de locomoción, la leve necesidad procesamiento (en comparación con otros sistemas) y la popularización del mismo en el mercado (existen numerosos UGVs de fabricación casera o relativamente barata). Por el contrario, existen situaciones en las que un vehículo con locomoción de ruedas no es lo bastante fiable y óptimo por lo que se debe investigar otra solución.

Los vehículos no tripulados de locomoción con patas tienen un alto coste en computación, unas bases físicas muy complejas y una corta vida en el mercado actual por lo que suelen quedarse relegados al segundo puesto dentro de las alternativas viables para dar solución a problemas de esta índole. Pese a esto, gracias a su locomoción con patas, proporciona una fiabilidad y capacidad de movimiento mayor que sus alternativas (Marc Raibert, 2008).

Centrados en la navegación, todos los robots no tripulados hacen uso de las técnicas de inteligencia artificial de alto nivel tales como algoritmos, sistemas complejos, toma de decisiones, etcétera, y es la perfecta aplicación de las teorías a un sistema práctico real.

Controladores.

Durante los últimos años se han utilizado diferentes técnicas o estrategias para el control de robots. Si bien es cierto que para los robots industriales es costumbre utilizar un controlador PID clásico, la elección de un tipo de controlador depende estrechamente de el tipo de robot que ha de utilizarse.

Por tanto, para elegir una estrategia de control correcta se han de tener en cuenta las características propias del robot que se posee.

Para empezar, se presentarán algunos términos y conceptos relacionados con la teoría del control, los sistemas de control y sus características. Sobre estas características se explicarán algunos controladores similares al controlador PID, protagonista en esta sección. Muchos de estos modelos son los cimientos de los siguientes. Esto significa que, basándonos en ellos, obtendremos los siguientes modelos, así como solventaremos sus carencias y problemas.

Más tarde hablaremos del controlador PID elegido, las razones de la elección y su relación con el problema.

Teoría de control, sistemas de control y características.

Lo primero que se ha de tener en cuenta a la hora de describir un controlador PID es: ¿qué entendemos por control? ¿qué es un sistema de control?

El término control tiene dos significados altamente relacionados: Primero, se refiere a la actividad de comprobar y probar que un dispositivo físico o matemático cumple con su comportamiento satisfactoriamente. Como complemento, el segundo significado, toma un significado más activo actuando e implementando decisiones que garanticen que esos comportamientos se desarrollan como se desean.

Podemos describir, entonces, que un sistema de control es aquel que trata de gobernar cierto proceso físico. Para gobernar dicho proceso físico se debe tener en cuenta los datos recogidos y los que se generan. Un sistema de control, por tanto, genera una salida a partir de una entrada.

Figura 4 - Esquema simple de un sistema de control

Para encontrar las ideas clave del control se puede tomar como referencia la Naturaleza, la evolución y el comportamiento de los seres vivos (Andrei, 2005).

La primera idea es la *retroalimentación*. Es esta característica, según las aportaciones de Charles Darwin (1805-1882), la responsable de la evolución de las especies. Esta idea ha sido trasladada a multitud de disciplinas y, como no podría ser menos, es la ingeniería (mas concretamente la robótica) ha tomado este concepto como estandarte en sus investigaciones (mediante las manos de los ingenieros de Bell Telephone Laboratory). Un proceso de retroalimentación es, por tanto, aquel en el que el estado del sistema, o su salida, determina la manera en que se comporta el controlador en cualquier instante.

La segunda característica clave es la necesidad de fluctuaciones. La idea, pese a parecer en un principio contradictoria, puede ser simple y efectiva. Básicamente consisten en que no es necesario centrar los esfuerzos en hacer que el sistema adquiera el estado deseado directa o inmediatamente. En muchas ocasiones ha resultado mucho más efectivo dejar que el sistema fluctúe hasta encontrar las

dinámicas que conduzcan al sistema al estado deseado sin forzarlo demasiado. Si bien este concepto fue bien acogido por economistas, como Hall [1907], los ingenieros se vieron reacios a esta técnica debido a sus evidentes carencias. Pero la necesidad de fluctuaciones es un principio muy general que encontramos hoy en día en la mayoría de los sistemas implementados por ingenieros y matemáticos.

El tercero y último concepto sobre el control teórico que presentaremos a modo de introducción es la optimización. Se entiende por optimización a la mejor manera de realizar una actividad. La mejor manera, en la mayoría de los casos, consisten en conseguir beneficios y reducir los gastos. Es una consecuencia obvia de todo sistema. La búsqueda del mayor beneficio con el menor de los costes sujeto a restricciones es materia de varias disciplinas y objetivo de sistemas de todo tipo, no solo en el campo de la ingeniería. Casi todos los problemas pueden ser reducidos a otros de lineales o quasi-lineales que impliquen la resolución de matrices de inecuaciones, dando lugar a la resolución de problemas de optimización.

Si unimos estos tres conceptos presentados y el modelo anteriormente descrito obtenemos lo que llamamos un sistema de control cerrado, donde la salida se compara con la entrada y se obtiene el error. El objetivo de nuestro sistema será hacer mínima esa señal de error.

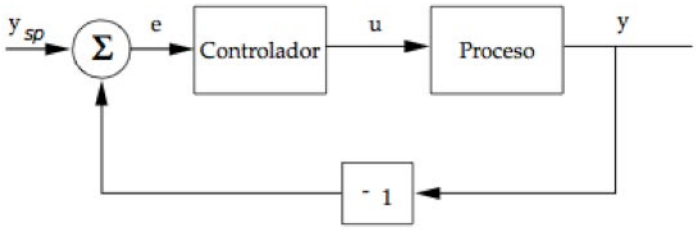


Figura 5 - Diagrama de un controlador

En la figura podemos encontrar dos componentes, el controlador y el proceso. La relación causal que existen entre estos componentes, las entradas y las salidas, se representan mediante las flechas. El proceso tiene una entrada conocida como variable de control. El proceso, que se encarga de tomar ese valor y realizar la acción concreta mediante un actuador (en nuestro caso los motores de las ruedas). Este proceso proporciona al sistema una salida y. El valor deseado por el sistema se denomina y_{sp}, valor de referencia. Por tanto, el error que se obtiene mediante la diferencia entre la salida obtenida y el valor de referencia sirven como entrada en la siguiente instante de tiempo del sistema.

Diferentes sistemas de control y sus principales características.

Una vez se ha explicado el concepto de sistema de control y sus ideas básicas se procederá a presentar distintos sistemas de control y, entre ellos, el control PID (Kart J. Aström, 2009).

Uno de los controladores más simples es el controlador *"on-off"*. Este control equivale a oscilar entre los valores de control máximos y mínimos siempre que el error resultante sea mayor o menor que cero. Este sistema tiende a oscilar y es común modificarlo con diferentes técnicas.

$$u = \begin{cases} u_{max}\,, if\ e > 0 \\ u_{min}\,, if\ e < 0 \end{cases}$$

El control proporcional es una posible solución a la oscilación del control *"on-off"*. Este se basa en que una pequeña variación en el error implicaría un cambio entre valores máximos. La idea principal de este controlador es aplicar una moderación al sistema anterior mediante una variable conocida como ganancia del controlador, *K*. El sistema no permite oscilar entre los valores máximos ya que *K* atenúa la señal u haciéndola proporcional al error.

$$u = K\left(y_{sp} - y\right) = K_e$$

Como se puede observar, el sistema anteriormente descrito puede generar una señal que se aleje demasiado del punto objetivo ya que la variable *K* es siempre la misma y no depende del error cometido anteriormente. Es decir, si el error que se está produciendo se va incrementando en el tiempo, puede que no se llegue nunca a alcanzar el objetivo.

Por tanto, es necesario mantener alguna relación entre la variable de control o ganancia del controlador *K*, y el error. Por tanto, se establece una relación integral de *K* con la integral del error.

$$u(t) = K_i \int_0^t e(\tau)\, d\tau$$

A este sistema se le conoce como control integral. En la figura anterior comprobamos que la ganancia integral k_i aumenta conforme mayor es el error cometido desde el principio hasta el instante *t*.

Adicionalmente se puede observar una característica especial que dota a este sistema la importancia que tiene. Si nos encontramos en un estado en el que el error es una constante, este solo dependerá de la variable *t*, por tanto existe una señal de control estacionaria y constante llamada u_0.

$$u_0 = k_i e_0 t$$

En el caso de que u_0 sea una señal constante implicaría que el error e_0 es 0, por lo tanto, u_0, como constante, siempre es 0.

Como resultado de estos dos sistemas mostrados anteriormente nace el controlador *PI*.

$$u(t) = Ke(t) + k_i \int_0^t e(\tau)d\tau$$

Como se ha descrito, si el error $e(t)$ es constante, solo debe ser cero así que se tendrá en cuenta la señal del controlador integral.

Por último, como apunte final al sistema, se le añade a este controlador la capacidad de anticiparse a la salida.

Controlador PID

La adquisición de la característica de anticipación a este sistema da como consecuencia el controlador *PID*.

Las siglas de este controlador son:

- *P*: Proporcional, como se explicó en secciones anteriores, referente al instante actual.
- *I*: Integral, referente al error cometido o errores acumulados, errores pasados.
- *D*: Derivativo, predicción del error que se va a cometer, errores futuros.

Esta anticipación es la extrapolación lineal del error.

$$T_d \frac{de(t)}{dt}$$

La expresión mostrada en la figura es la predicción lineal del error T_d unidades de tiempo en el futuro.

$$u(t) = K\left(e(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau + T_d \frac{de(t)}{dt}\right)$$

O su homónima:

$$u(t) = K_p \cdot \epsilon(t) + K_i \cdot \int_0^t \epsilon(t) \cdot d\tau + K_d \cdot \frac{d}{dt}\epsilon(t)$$

Por tanto, es un sistema que toma como parámetros la ganancia proporcional K, el tiempo integral T_i o acumulativo y el tiempo derivativo T_d o predictivo.

Si tomamos como referencia la figura anterior los parámetros a tomar en cuenta son K_i, K_d y K_p.

Dependiendo de la configuración del estos tres parámetros únicos que recibe el controlador, conocidos como ganancias, se determinará el comportamiento de este controlador.

La regulación de la ganancia proporcional K_p mide la agresividad del controlador, generando una señal más alta y, por lo tanto, provocando grandes cambios en la señal de salida.

El parámetro K_i, ganancia integral, elimina, como se explicó antes, los errores estacionarios pero una sobre actuación de este parámetro provoca una oscilación inicial alta.

Por último, la configuración del parámetro K_d reducen las oscilaciones iniciales pero provocan que el sistema sea más lento. Además, este parámetro es muy sensible a errores puntuales debidos al ruido de la señal, por lo que hacen al sistema muy inestable.

La configuración dará como resultado los ensayos de este proyecto, los cuales se explicarán posteriormente.

El sistema de control *PID* es utilizado en un gran parte de los problemas de control que se pueden encontrar. El aumento de complejidad o la implementación de sistemas más complejos que el controlador *PID* difieren de este por la sofisticación de la predicción.

Pseudocódigo del controlador PID

Durante este apartado se explicará brevemente como debe ser una implementación del controlador PID deseado. (Wescott, 2000)

Se dividirá en tres pasos (uno por cada componente del controlador) más un resultado final y otro de adecuación del error. La entrada no deberá ser más que el dato actual y el deseado.

1. *error = entradaDeseada - entradaActual*
2. *TérminoP = GananciaP * error.*
3. *EstadoI = EstadoI + error*
   Si (EstadoI > EstadoIMáximo) Entonces EstadoI = EstadoIMáximo
   Si (EstadoI < EstadoIMínimo) Entonces EstadoI = EstadoIMínimo
   *TérminoI = GananciaI * EstadoI*
4. *TérminoD = GananciaD * (error - EstadoD)*
   EstadoD = error
5. ***Resultado = TérminoP + TérminoI – TérminoD.***

Como se puede observar los términos de este controlador corresponden a los componentes del controlador. Además, las ganancias son las variables *K* del controlador.

Este controlador es un controlador sencillo y, por lo tanto, puede ser mejorado alterando este pseudocódigo.

Sensores ópticos, control visual.

"*Una de las empresas más ambiciosas consiste en dotar a los ordenadores de la facultad de relacionarse con su entorno del mismo modo que lo hacen los humanos: a través de los sentidos*" (Ana González Marcos, 2006)

La visión, tanto como para el ser humano como para el ordenador, se basa principalmente en dos fases: captación e interpretación.

Si bien a primera vista la captación podría resultar un asunto muy complejo, hace mucho tiempo que este problema esta resuelto en la visión artificial: el "ojo" del ordenador es su cámara web y el sensor óptico la retina.

Una vez extraída esta imagen queda, nada más y nada menos, que la interpretación. Se llama interpretación al proceso de extraer información sobre objetos, distinguir entre ellos y extraer información.

Por lo tanto, se podría definir a la visión artificial como "*la deducción automática de la estructura y propiedades de un mundo tridimensional, posiblemente dinámico, mediante una o varias imágenes bidimensionales de este mundo".* (Ana González Marcos, 2006)

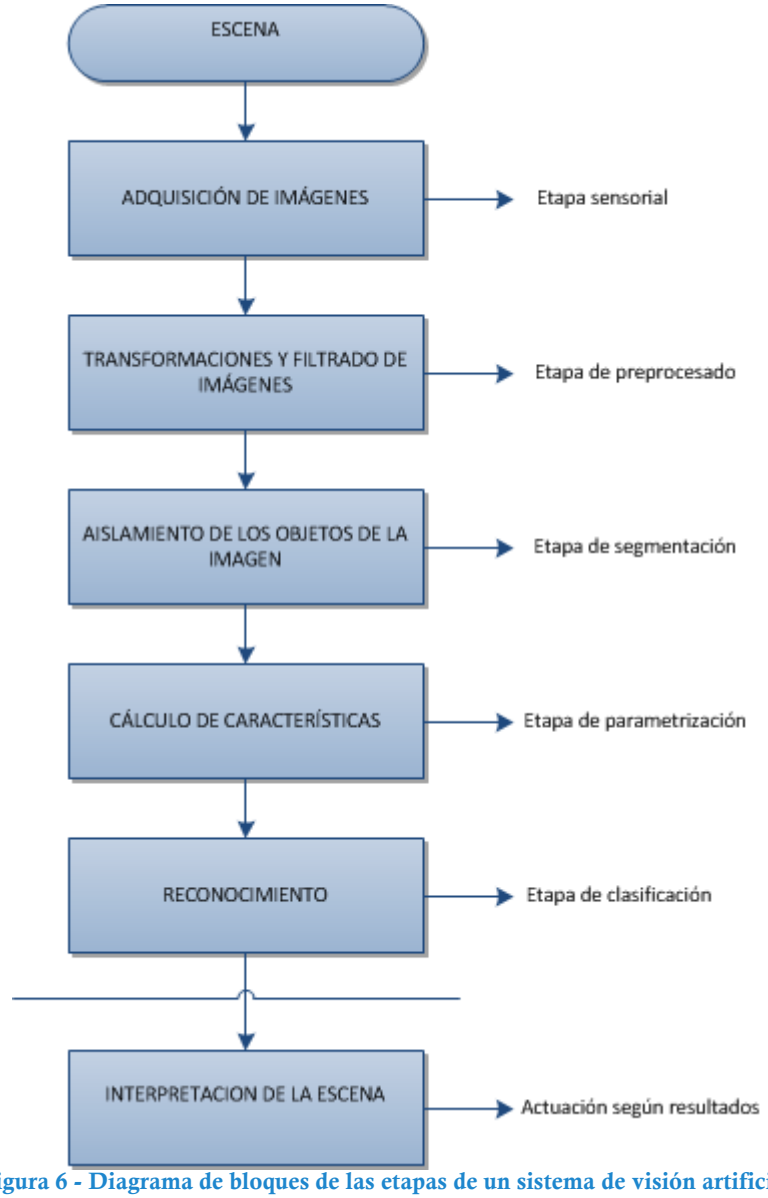


Figura 6 - Diagrama de bloques de las etapas de un sistema de visión artificial

Etapa de adquisición

En la visión artificial, los dispositivos de captación de información reciben el nombre de cámaras que, mediante el uso de la óptica, produce una imagen completa del dominio una o varias veces en función del tiempo.

Un sensor óptico se basa en el aprovechamiento de la interacción entre la luz y la materia para determinar las propiedades de esta.

Figura 7 - Diagrama de detección con un sistema óptico

Tal como presenta el diagrama, a raíz de una fuente de luz se extrae una muestra mediante el sistema óptico. Una vez extraída la muestra se dispone a transformar esa información en datos relevantes para el sistema de manera que sean procesables.

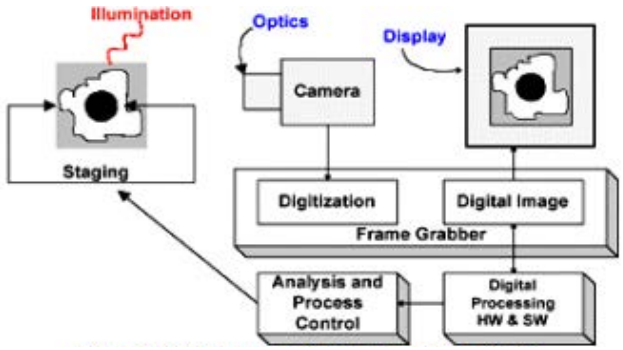


Figura 8 - Diagrama de bloques de un sistema SVA

Una visión algo más completa del diagrama anterior puede definirse de la siguiente manera:

1. El proceso de captación comienza cuando la óptica recibe información analógica del mundo real tridimensional y la transforma a una imagen estática bidimensional analógica.
2. Dicha imagen sufre un proceso de digitalización que la transforma a una relación funcional entre posicionamiento dentro de la imagen y la luz.
   Esta luz es presentada mediante dos componentes que son la luz incidente que procede de la escena y la luz que reflejan los objetos de dicha escena. Ambos componentes *i(x,y)* y *r(x,y)* conocidos como iluminación y reflectancia, se combinan como producto dando lugar a *f(x,y)*. Por lo tanto podríamos definir una imagen con la siguiente función:

$$f = \begin{bmatrix} f(1,1) & \cdots & f(1,M) \\ \vdots & \ddots & \vdots \\ f(N,1) & \cdots & f(N,M) \end{bmatrix}$$

Etapa de transformaciones y filtrado de imágenes

Una vez recibida la señal del sensor óptico, la labor del sistema de detección es modificar dicha señal para extraer la información.

Para ello se realiza la acción de preprocesado y más tarde se hace uso de diferentes técnicas y transformaciones que permitan reconocer objetos.

El preprocesado consiste en operaciones básicas y geométricas sobre la imagen, así como filtrados y operaciones sobre el histograma de la imagen produciendo una nueva que repare los desperfectos producidos por la óptica o el ruido de la escena y ayudando a la obtención de objetos de la misma.

Algunas de estas operaciones básicas son:

- Inversión: obtener el valor inverso de cada pixel o muestra.
- Operaciones aritméticas entre imágenes y/o constantes. Algunas son el producto, la adición, substracción…
- Complementación: La obtención del negativo de la imagen invirtiendo el valor de los pixeles de la imagen.
- Transformaciones no lineales: como la corrección gamma, afilado de la imagen…

Algunas de las operaciones geométricas son:

- Algoritmos de traslación o desplazamiento: sustituir el pixel por el correspondiente a sus coordenadas más un desplazamiento.
- Algoritmos de rotación: Desplazar el pixel por el correspondiente en a sus coordenadas más un giro desplazador.

Algunos filtros utilizados son:

- Normalización
- Filtro de media o suavizado
- Filtro detectores de bordes mediante gradientes con promediado

Algunas operaciones sobre el histograma de la imagen son:

- Función cuadrado: de cada valor del histograma, la obtención de su valor al cuadrado.
- Función cúbica: similar a la anterior pero obteniendo el cubo.
- Ecualización del histograma: Mejora del contraste de la imagen obteniendo más diferencia ante una gran cantidad de puntos en torno a un nivel gris.

Figura 9 - Imagen sin ecualización del histograma

Figura 10 - Imagen con ecualización del histograma

Una vez realizado el preprocesamiento de la imagen, se procede a obtener una segmentación para obtener los objetos separados en la misma imagen.

La segmentación consiste en dividirla en zonas disjuntas para diferenciar diferentes objetos.

Para ello se hace uso de diferentes transformaciones tales como las morfológicas, para luego utilizar diferentes técnicas de segmentación tales como las basadas en umbrales o en bordes.

Algunas de las transformaciones morfológicas son:

- Dilatación binaria
- Erosión binaria
- Apertura
- Cierre
- Esqueletos.

Algunas de las técnicas de segmentación basadas en umbrales son:

- Umbralización en búsqueda de mínimos.
- Umbralización basada en reconocimiento de formas.

Algunas de las técnicas de segmentación basadas en bordes son:

- Relajación de bordes.
- Extracción de frontera.

Una vez realizadas estas operaciones se procede a analizar la imagen obtenida. Es entonces cuando se obtienen los objetos concretos de la imagen.

Para este análisis se hace uso de técnica como representación de fronteras o representación de regiones que delimitan los objetos para su posterior uso en la situación del vehículo.

Interpretación de la señal: medida de distancias y ángulos.

Una de las mayores problemáticas a la hora de ajustar este tipo de herramientas a las soluciones de cualquier proyecto es la interpretación de los datos. En este apartado se explicará como averiguar el ángulo de giro dado un objeto identificado en la imagen.

Para ello será necesario conocer el ángulo de visión que posee la cámara. Este ángulo determina qué parte de la escena es captada por el sensor de la cámara.

Figura 11 - Campo de visión

Como se puede observar en la figura anterior, el campo de visión de la cámara es mucho más reducido que la escena total y, por lo tanto, es posible obtener una medida exacta del ángulo máximo que la cámara puede captar.

El ángulo de visión viene determinado por la distancia horizontal que el sistema es capaz de abarcar.

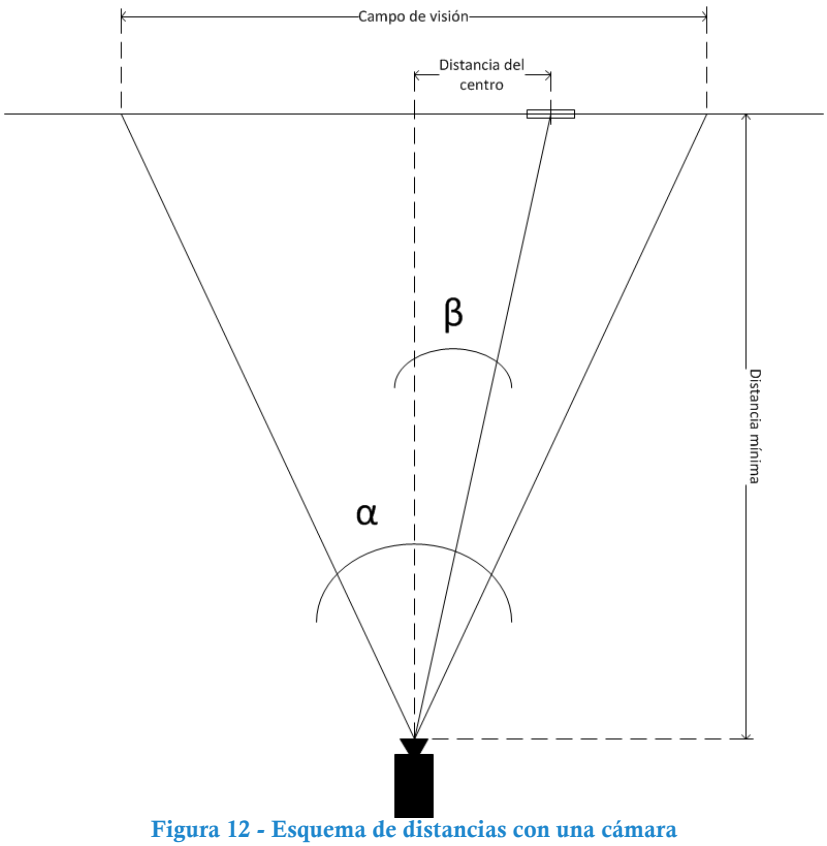


Figura 12 - Esquema de distancias con una cámara

Como se puede observar en la *Figura 12,* el ángulo α es siempre el mismo a pesar de que la cámara esté en movimiento. Si bien es diferente la porción de escena que refleja el sistema el ángulo de dicha porción será siempre α. Si se coloca un objeto en la imagen que pueda ser detectado por el sistema se podrá obtener la distancia que existe desde el centro de la escena hasta el centro del objeto obteniendo una medida de giro que se ha de realizar para colocar dicho objeto en el centro de la escena.

Tal y como muestra la *Figura 13*, el objeto se encuentra a cierta distancia del centro. Dicha distancia genera un nuevo ángulo β que puede ser calculado mediante el campo de visión y su ángulo (tal y como se puede interpretar en la *Figura 12*).

Figura 13 - Objeto en la imagen

Para averiguar cual es exactamente el ángulo de visión es necesario medir la distancia mínima. Por ejemplo, mediante un objeto de medida estándar (por ejemplo un folio de 30 cm A4) se obtiene dicha distancia acercándolo o alejándolo de la cámara hasta que abarque todo el campo de visión.

Una vez obtenida dicha medida, conocemos entonces su ángulo

$$tg\ \alpha/2 = \frac{30/2}{distancia\ mínima} \rightarrow \alpha = 2\ arctg\ \frac{30/2}{distancia\ mínima}$$

Tal y como se ha averiguado α el paso para obtener β es similar.

$$tg\ \beta = \frac{distancia\ del\ centro}{distancia\ mínima} \rightarrow \beta = arctg\ \frac{distancia\ del\ centro}{distancia\ mínima}$$

Por lo tanto habiendo obtenido la *distancia mínima* es posible localizar objetos en una imagen y el ángulo de giro para posicionarlo en el centro.

Descripción del sistema

Una vez presentadas las bases teóricas del proyecto a desarrollar se explicará de manera sencilla y concisa el objetivo de este proyecto, sus componentes y su finalidad.

El sistema a implementar simula el control de un vehículo no tripulado terrestre utilizando dispositivos simulados y en un entorno también simulado. El sistema debe mantener un movimiento automático cumpliendo ciertos objetivos o superando unas metas establecidas. El sistema debe ser flexible para poder implementar cualquier conjunto de objetivos o metas y dando facilidad a la inclusión de mejoras en el sistema de control. La realidad ubica dicho sistema con dispositivos reales con tasas de error en la extracción de datos y problemas de integración entre los distintos dispositivos, por lo tanto este proyecto debe alcanzar el nivel de compromiso de implementar, al menos, el manejador de un dispositivo real y probar su funcionamiento con el software de control.

Durante las siguientes páginas, este documento describirá el entorno de desarrollo y ejecución necesario para alcanzar los objetivos anteriormente citados haciendo uso de las técnicas teóricas explicadas en los apartados previos al presente. Estas técnicas son explicadas con detalle en dichos apartados pero, a continuación, se enumerarán dando, como información adicional, una breve argumentación de su selección para este proyecto.

- Dispositivos ópticos: El dispositivo real a implementar será una cámara web integrada con el robot simulado orientada hacia el frontal del mismo. Esta cámara transmite datos reales que serán procesados y analizados para extraer la información necesaria para manejar el sistema de control.

  Se ha elegido la cámara web por considerarse uno de los dispositivos reales más cercanos a la comprensión humana y, a la vez, debido a su familiaridad con la fase de desarrollo. Además, para la implementación de manejadores del dispositivo se disponen de múltiples herramientas útiles con las que procesar y analizar la información que presta este dispositivo. El objetivo de usar este dispositivo es el de conseguir obtener datos de control coherentes y útiles únicamente con esta cámara web.

- Controladores: Si bien muchas de las técnicas de control son muy comunes en el mundo de la robótica, el *Controlador PID* ha sido el elegido por su sencillez de implementación y sus resultados altamente satisfactorios. Además esta técnica de control abarata los costes computacionales dando una solución aceptable para el problema perseguido (si bien existen diversos controladores que proporciones soluciones óptimas, esta técnica resulta eficaz y eficiente frente a las demás).

- Sistema simulado: Es inviable pensar en una implementación sobre robots reales sin antes desarrollarlo en entornos virtuales. Es necesario para la viabilidad del proyecto (económicamente) y para la interpretación de los resultados previos al salto al entorno real.

  Durante este proyecto se implicará, además del sistema simulado, un sistema híbrido entre la simulación del entorno elegido y el dispositivo real implementado sirviendo como base a futuras líneas de trabajo en ese sentido.

- Sistema ampliable: El proyecto deberá implementar un sistema fácilmente escalable y con una línea de trabajo futura amplia y manejable para futuros desarrolladores. Por ello, la conjunción de las anteriores técnicas deben estar acompasadas con el requisito de la mayor flexibilidad posible en el sistema.

Visión global del sistema

Por último se decide añadir una imagen global del sistema y una breve explicación para que el lector reconozca los principales componentes del sistema y el lugar que ocupa el desarrollo que más tarde se expondrá.

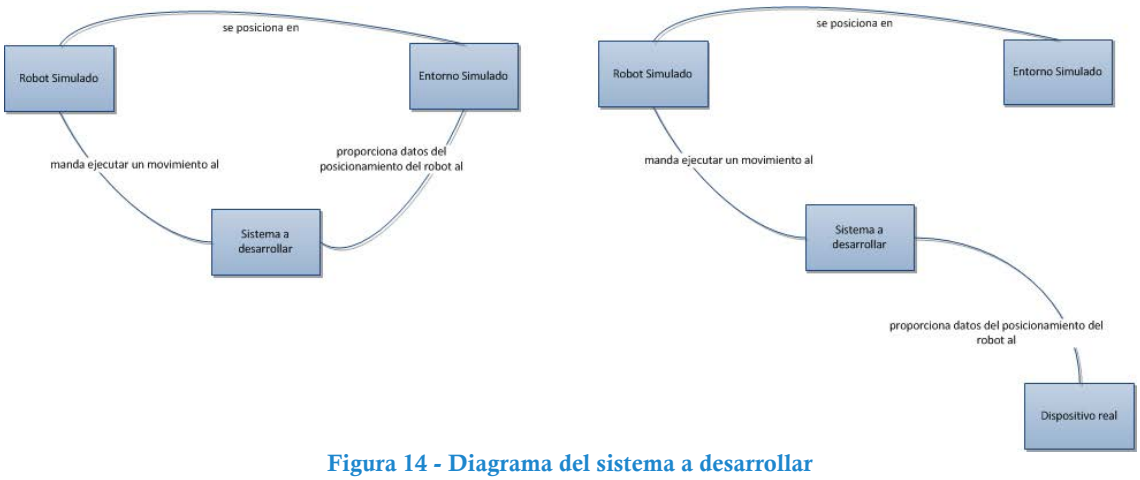


Figura 14 - Diagrama del sistema a desarrollar

Como se puede apreciar en la imagen, el sistema a desarrollar deberá contener la suficiente autonomía para decidir cual será el siguiente movimiento. Tras decidirlo, le ordenará al robot simulado que lo ejecute y este lo hará sobre el entorno simulado. En el caso de tener un dispositivo real, la funcionalidad es la misma y el sistema deberá transformar los datos del dispositivo en un movimiento del robot.

Como se puede comprobar todos los componentes de este sistema están sin solapar y, por lo tanto, se pueden modificar o cambiar dando paso a nuevas implementaciones en entornos no simulados y robots reales.

El sistema simulado ejecutará movimientos siguiendo como objetivos distintos puntos en el mapa establecidos anteriormente. Estos puntos serán llamados *trayectoria*.

En el sistema implementado con un dispositivo real se realizará el movimiento con el objetivo de perseguir el objeto detectado. Al llegar a una posición cerca de dicho objetivo el sistema se detendrá hasta que este vuelva a estar lo suficientemente lejos.

Entorno

Este aparado del documento servirá para encuadrar el trabajo realizado bajo un marco delimitado por el entorno de trabajo. En el se explicarán las diferentes herramientas utilizadas durante el desarrollo del proyecto y sus posteriores pruebas así como las que el usuario final deberá utilizar para su ejecución.

Pese a existir muchas herramientas para el desarrollo de entornos virtuales de simulación de robots se ha elegido Player y Stage por pertenecer a las implementaciones de proyectos anteriores y por ser las herramientas más sencillas en el campo de la simulación. Además, la comunidad de colaboradores alrededor del mundo es muy amplia y se han podido solucionar la mayor parte de los problemas encontrados durante su desarrollo gracias este valor añadido.

El entorno de trabajo es un factor muy importante a la hora de realizar el proyecto ya que muchas de las dificultades o facilidades que se pueden encontrar en estos casos suelen ser causadas y a su vez solucionadas por el mismo. La falta de un entorno adecuado, o incluso una mala elección de las herramientas (como por ejemplo la no elección de una herramienta) puede provocar que se realice varias veces el mismo trabajo sin percatarse de que este ya ha sido realizado con anterioridad.

Otras alternativas (como *OpenRDK* o *Gazebo*) son más completas que las elegidas pero se decidió utilizar las mencionadas porque el proyecto no necesitaba un sistema muy complejo sobre el que implementarse, si no uno simple que permitiese adecuar su solución.

De un modo abstracto, el proyecto necesitaba de tres cosas:

- Un simulador del entorno donde se puedan colocar robots.
- Un simulador de robot que contenga los principales sensores.
- Una interfaz de comunicación entre ellos muy sencilla.

Este proyecto se sitúa entre los sensores del robot y el entorno.

Además de esto, el proyecto deriva a una implementación paralela con un dispositivo óptico no simulado para extraer información de él y usarla en el sistema. El manejo de dispositivos ópticos sin una herramienta específica resulta harto difícil y por eso se ha elegido la plataforma *OpenCV* para desarrollar sobre ella. Esta plataforma es muy completa y sencilla de utilizar además de ser la más utilizada en el mundo de la investigación y desarrollo por lo que el factor comunitario del mismo es altamente positivo.

Si bien no se tratará de explicar a fondo el funcionamiento de cada uno de los entornos, se creará una visión global y específica a grandes rasgos de cada uno de ellos para que el lector comprenda la labor y el por qué de su uso.

Player.

Player es una capa de abstracción entre el hardware del robot y el software del cliente. Se podría definir como un servidor que, corriendo en el robot, proporciona una interfaz simple y limpia para los sensores y actuadores bajo el protocolo de red IP. Mediante esta interfaz, los programas cliente se comunican con Player a través de sockets TCP leyendo datos de los sensores y escribiendo comandos a través de los actuadores y configurando los dispositivos durante la ejecución (Reed Hedges, Player Project, 2008).

Player proporciona, mediante una arquitectura cliente-servidor, una manera simple y transparente de comunicar el código cliente con los dispositivos del robot.

La abstracción conseguida se debe a que el servidor se comunica con el robot mediante el uso de drivers específicos de los dispositivos pero estándares. La mayoría de los fabricantes son conscientes de que Player es una plataforma muy extendida y han desarrollado sus drivers correspondientes con ella para facilitar el trabajo a los desarrolladores.

Gracias a esta abstracción, Player proporciona una interfaz para todos los dispositivos del tipo al que hacen referencia. Un ejemplo: Un dispositivo de cámara, que proporciona un flujo de datos con las imágenes que esta capta durante un periodo de tiempo, puede ser utilizada en una arquitectura con Player y Stage. Resulta evidente que existen muchos fabricantes de cámaras no todas tienen las mismas especificaciones y componentes. Pero también es obvio que la función de dicha cámara es proporcionar imágenes. Es en este punto donde Player aprovecha el punto en común de todos los dispositivos del tipo "cámara". Al usuario final le es irrelevante la cámara que esté utilizando (más allá de las especificaciones técnicas de la cámara relevantes para el problema que se trata de abordar), por lo tanto, proporcionar una interfaz común para todas estas cámaras es, sin duda, un capa de transparencia del hardware muy útil para el desarrollador.

Además, durante el transcurso de este proyecto, no se dispuso de los dispositivos reales de un robot real en todo momento por problemas de localización y de disponibilidad (ya que varias personas y departamentos trabajaron con los mismos dispositivos durante el mismo intervalo de tiempo). Gracias a este entorno simulado se pudo avanzar la mayor parte del tiempo sin necesidad de poseer los dispositivos reales que, finalmente, se utilizarían y, en su lugar, se podía usar otros del mismo tipo. Volviendo al ejemplo de la cámara, durante la implementación del módulo de este proyecto para este dispositivo, se utilizó una cámara web estándar y así no se impone gran dificultad al trasladar este sistema a la cámara IP que el robot posee.

Player es una plataforma independiente de arquitectura. Esto implica que es posible desarrollar bajo cualquier lenguaje o máquina siempre que esta disponga de una conexión de red.

Una de las ventajas de utilizar una plataforma como Player, que proporciona una interfaz única para distintos fabricantes de sensores y dispositivos, es la posibilidad de portar el mismo programa cliente entre varios robots o arquitecturas. Esto proporciona una útil plataforma de desarrollo bajo un entorno virtual que ha sido esencial durante el desarrollo de este proyecto. Desarrollar el programa cliente en simulado para, más tarde, trasladarlo al robot real no supone ningún problema de compatibilidad entre arquitecturas.

Además, Player proporciona un soporte para cualquier número de clientes. De esta manera, desde la misma plataforma, se pueden controlar distintas arquitecturas y comunicarse entre ellas. Gracias a esta característica se puede, por ejemplo, compartir datos entre distintos terminales robot para completar tares conjuntamente lo que es, sin duda, una característica muy interesante para llevar a la práctica proyectos como las estrategias cooperativas paralelas para la solución de problemas de optimización (Carlos Cruz Corona, 2007) o la investigación de la aun joven, inteligencia artificial distribuida.

El hecho de recibir lecturas o datos de varios autómatas, trabajando independientemente y colaborando entre ellos utilizando una plataforma única como Player es una ventaja a la hora del desarrollo de funciones altamente útiles.

Por último, cabe destacar que Player es una plataforma enorme de software libre. Además de ser gratuita y tremendamente colaborativa, dispone de un soporte proporcionado por desarrolladores que trabajan, también, sobre esta plataforma.

Stage.

Stage es un software de simulación de autómatas móviles, sensores y objetos en un entorno bidimensional relacionado con mapas de bits. Al contrario de otros simuladores como *Gazebo*, *Stage* no contempla la posibilidad de emular fielmente dispositivos o robots móviles, si no que trata de dar soporte a múltiples dispositivos sin que esto repercuta en el rendimiento del equipo que esta ejecutando dicha simulación.

Gracias a esto y a que *Stage* es una plataforma simple y altamente escalable, desarrollar y probar los programas clientes encuentran esta solución muy apropiada ya que no son necesarias arquitecturas con alto nivel computacional. Los autómatas, además, son compiladas y cargadas durante tiempo de ejecución y pueden ser añadidas a cualquier modelo. Los controladores, además, tienen acceso completo a la API de Stage. (Reed Hedges, Stage Project, 2010)

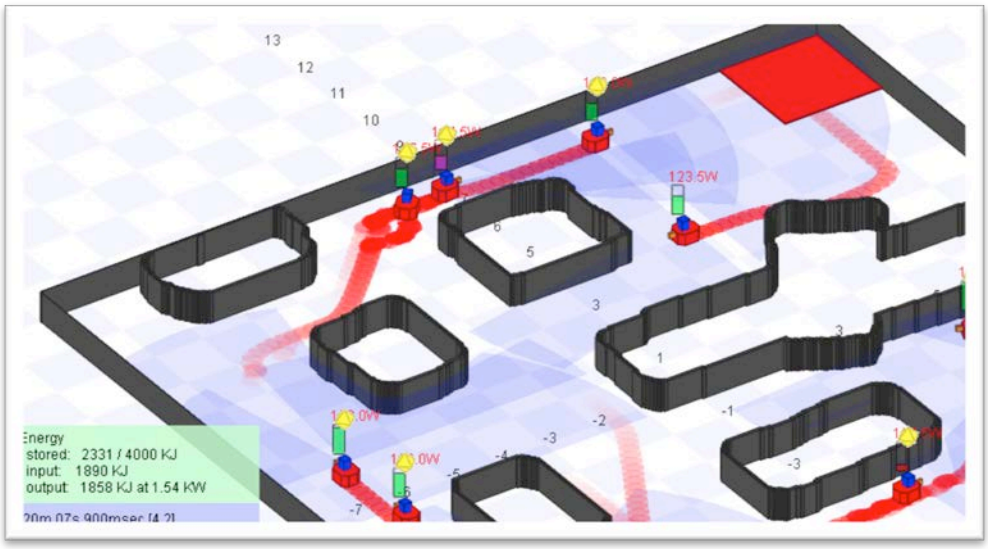


Figura 15 - Ejemplo de Player con Stage

Integración de Player y Stage.

El uso combinatorio de estas dos tecnologías proporciona un entorno simulado óptimo para los propósitos de este proyecto. Mediante la librería "*libstageplugin*" los usuarios pueden escribir programas con controladores o algoritmos para los sensores y ejecutarlos como clientes del servidor de Player. El uso de esta plataforma combinada recrea los autómatas simulados de una manera tan fiel a la realidad que, a la hora de trasladarlo al entorno real, no necesita cambios (o en su defecto, estos son mínimos).

Además, con estas técnicas, el desarrollador no tiene por que disponer físicamente de los dispositivos a utilizar porque la mayoría de los disponibles actualmente en el mercado pueden ser simulados por Stage.

Además, Stage proporciona una librería en C++ que puede ser integrada en los programas del usuario y así no depender de un simulador como Player.

Librerías OpenCV.

OpenCV son unas librerías de código abierto que implementan funcionalidades relacionadas con la visión artificial (Gary Bradski, 2008).

Las librerías *OpenCV* están escritas en C y C++ y corren bajo sistemas operativos Linux, Windows y Mac OSX. Su implementación puede hacer uso de múltiples procesadores y tiene como objetivo la eficiencia y robustez sobre aplicaciones en tiempo real. Uno de sus principios siempre ha sido proveer al usuario de una infraestructura sencilla para desarrollar aplicaciones con componentes de visión artificial de manera rápida y satisfactoria.

Además, las librerías *OpenCV* ofrece librerías completas de aprendizaje automático ya que, en la mayoría de los casos, la visión artificial y esta van de la mano.

Estas librerías son utilizadas por software médico, de seguridad, inspección de productos… Un ejemplo claro de la extensión de su uso es la empresa Google para su software Google Street View.

Open CV lanzó su primera versión estable en 2006 y, a día de hoy, cuenta con la versión 2.3.

Funcionamiento.

Las librerías Open CV están estructuradas en 5 componentes básicos:

- Componente *CV*: Procesamiento de imagen y algoritmos de visión.
- Componente *MLL*: Procesamiento estadístico y herramientas de cluster.
- Componente *HighGUI*: GUI e interfaces de Video e Imagen.
- Componente *CXCore*: Estructuras básicas y algoritmos, soporte XML y funciones básicas de dibujo.
- *CVAux*: Cubre aspectos muy variados tales como modelos ocultos de Markov, visión estereoscópica, clases de calibración de cámara en C++…

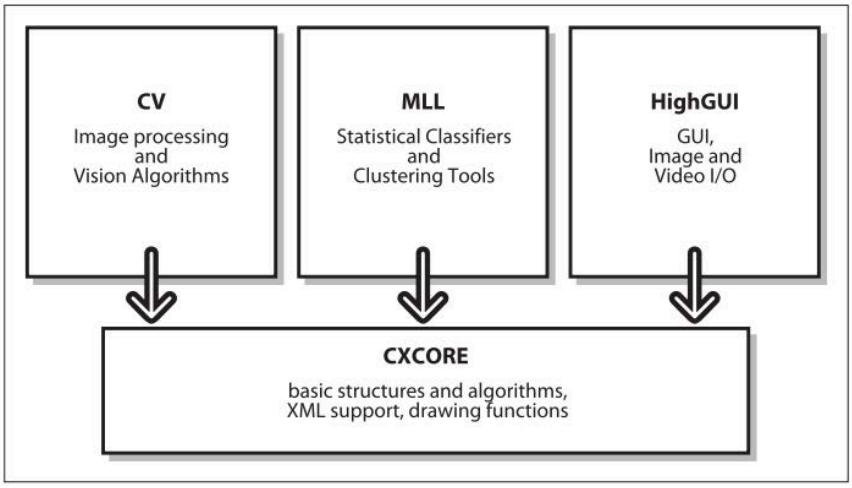


Figura 16 - Componentes de las librerías OpenCV

Durante este proyecto se acotarán muchas de sus funcionalidades por resultar irrelevantes para la implementación del componente óptico que se utilizará.
Como se explicó anteriormente, el campo de la visión artificial tiene dos fases claramente diferenciables: la captación de señal y el análisis de la misma.

Ambas fases son abordadas con esta librería dado que su excelente funcionamiento y sencilla interfaz las hace accesibles para cualquier tipo de cámara y desarrollador no experto en los campos de la visión computacional.

Captación de la señal.

La captación de la señal se realiza mediante las funciones de captura siguiendo el siguiente patrón:

Captura = operaciónDeCaptura()

Una vez almacenada la variable de la captura, se utilizará las funciones de captación de frames de esa variable para almacenarlos y poder tratarlos de manera individual.

Frame = operacionCaptacionFrame(captura)

De esta manera, la señal está siendo recogida y muestreada por frames para su posterior transformación.

Análisis de la señal.

Una vez realizada la captación de la señal y habiéndola divido en frames, será necesario realizar transformaciones que conviertan una imagen con ruido y degradada en una con las características necesarias para el reconocimiento de objetos.

Para ello, se dispone de varias funciones de transformación tales como el suavizado, cambio de tamaño, rotación, aplicación de filtros, transformación en escala de grises, reconocimiento de contornos, …

Tras aplicar las transformaciones necesarias puede disponer de algoritmos de aprendizaje automático que, tras pasar las imágenes sin ruido, reconozcan objetos en la imagen.

Algunas de estos algoritmos son:

- Algoritmos de K-medias
- Distancia de Mahalanobis
- Clasificadores Naïve/Bayes
- Árboles de decisión binarios
- Detección de caras (Haar Classifer).

Para este proyecto se utilizará detección de caras.

Más adelante, este documento explicará la implementación de manera más exhaustiva del uso de esta técnica y su aplicación en el proyecto.

Desarrollo del software

Este capítulo del documento presenta uno de los puntos más importantes del proyecto. El desarrollo del software explicará el cómo se ha realizado el software bajo los siguientes puntos: sus requisitos, su punto de partida, el diseño de la solución y su correspondiente implementación.

Durante el trascurso de cada una de estas fases se ha modificado una y otra vez el diseño y la implementación del software pero, en este documento, solo se mostrará el resultado final acompañado de una breve explicación sobre las decisiones tomadas para entender de manera global el objetivo del software y sus requisitos.

Si bien el objetivo de este proyecto es la implementación de un sistema que pueda manejarse para realizar pruebas relevantes y demostrar prácticamente la posibilidad de implementar un sistema de control que conlleve un dispositivo real, no se ha de dejar atrás la otra gran necesidad de este proyecto que es la escalabilidad.

Durante las siguientes páginas el lector se encontrará con dos versiones del software: la simulada y con el dispositivo real. Para hacer más entendible este documento, tras la descripción global del software, se realizará primero la disertación sobre el simulado y más tarde sobre el dispositivo.

Principales objetivos: ¿Qué se desarrolla?

El principal objetivo de este proyecto es crear un software que permita al usuario realizar pruebas con un controlador ya implementado o con futuros controladores. Además, es necesario poner en práctica la implementación de un dispositivo real y probarlo con el sistema.

Esto conlleva a dos objetivos principales:

- Crear un software de control útil.
- Diseñar un entorno de desarrollo altamente escalable.

Que el sistema deba de ser altamente escalable significa que esté lo más abierto y accesible posible para futuras implementaciones no dependiendo del entorno, de las plataformas o del sistema a implementar. En un futuro, este sistema puede emplearse para desarrollar otros controladores, utilizar el ya implementado, implementar otros dispositivos… Por lo tanto, se debe establecer unas bases muy ligeras y fáciles de interpretar para que sea lo más moldeable posible.

El software de control ha de ser útil. Esto significa que debe poderse realizar muchas pruebas de manera rápida y eficiente. A ser posible sin realizar cambios en el código. Para ello se ha implementado una versión en el sistema simulado que es acompañado de una interfaz gráfica de usuario donde este podrá introducir los datos necesarios para realizar pruebas en el sistema simulado o en la cámara.

Punto de partida.

El punto de partida de este sistema es otro sistema que pueda recibir los datos de los sensores al más alto nivel posible. Además, requiere que el sistema esté desarrollado en unas plataformas específicas y que pueda ser modificado en caso de necesitarlo.

Estas plataformas son *Player/Stage* y el software en el que se ha basado este sistema es el proyecto (Rebollo, 2010).

Si bien el sistema de Tomás es un software central que se comunica con los dispositivos simulados o reales del sistema, este proyecto se basa en una modificación del mismo y una posterior ampliación abarcando otra capa de software más alta.

En esta capa se pueden realizar pruebas con un controlador *PID* implementado o incluso implementar otro para su posterior uso.

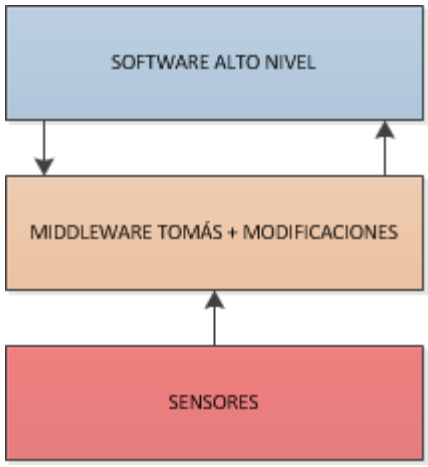


Figura 17 - Entorno de desarrollo

Si bien el proyecto de Tomás era una capa de abstracción más a *Player*, este software tiene como principal objetivo añadir horizontalmente más funcionalidad (con la entrada de un controlador *PID*) y abstraerse más en el desarrollo de herramientas no relacionadas directamente con *Player* (por ejemplo, otros controladores, otros objetivos diferentes a el paso por trayectorias…).

La iteración de nuestro con *Player/Stage* puede verse claramente en el siguiente gráfico.

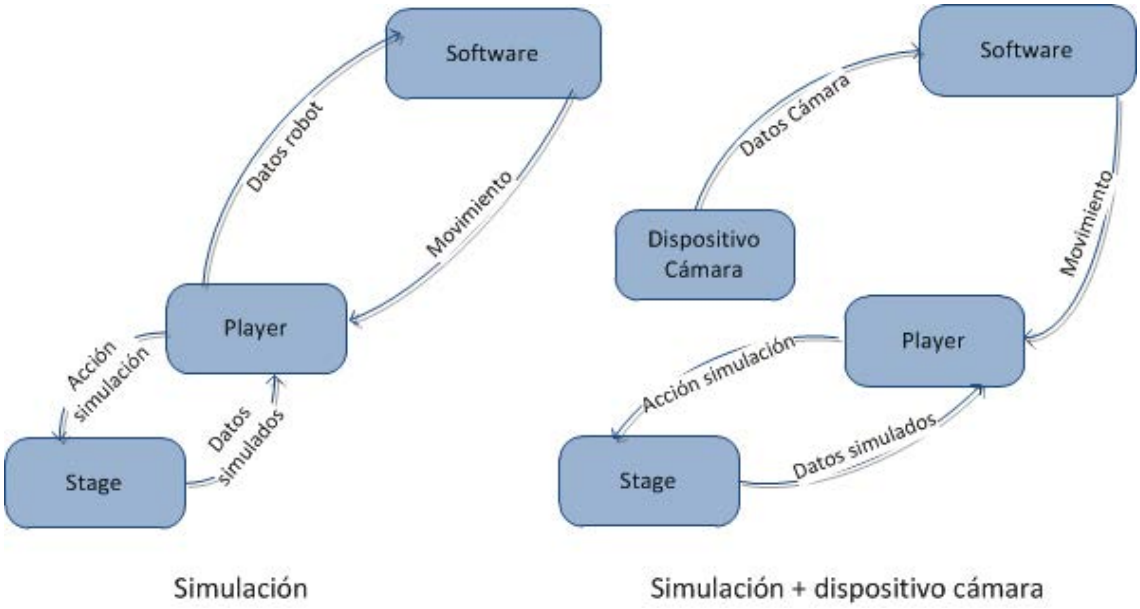


Figura 18 - Iteración del software con *Player / Stage*

El sistema implementado se sitúa entre los datos simulados por *Stage* y el software del *Robot* de Stage. De esta manera, el sistema corrige la trayectoria del robot incidiendo en el movimiento a realizar y lo sesga o lo pronuncia dependiendo de la decisión del controlador.

Diseño del software.

El software implementado ha de ser lo suficientemente escalable como para que futuras implementaciones puedan ser realizadas a expensas de las ya hechas. De esta manera los usuarios que quieran implementar sus propios controladores, sus capas superiores o cualquier otro dispositivo que no sea la cámara, tendrá libertad de movimiento con el sistema.

Para ello se ha optado por diseñar tres módulos claramente diferenciables, el Filtro, el Controlador y el Estado y un cuarto que contiene el bucle principal llamado Movimiento.

A continuación se describirá cada uno de los componentes basándose en su funcionalidad.

Filtro

El *Filtro* se encarga de interactuar con el controlador y el estado sirviendo, como su propio nombre indica, de filtro entre el bucle principal y el estado. El *Filtro* tiene como función principal el cálculo del siguiente movimiento que ha de realizar el robot.

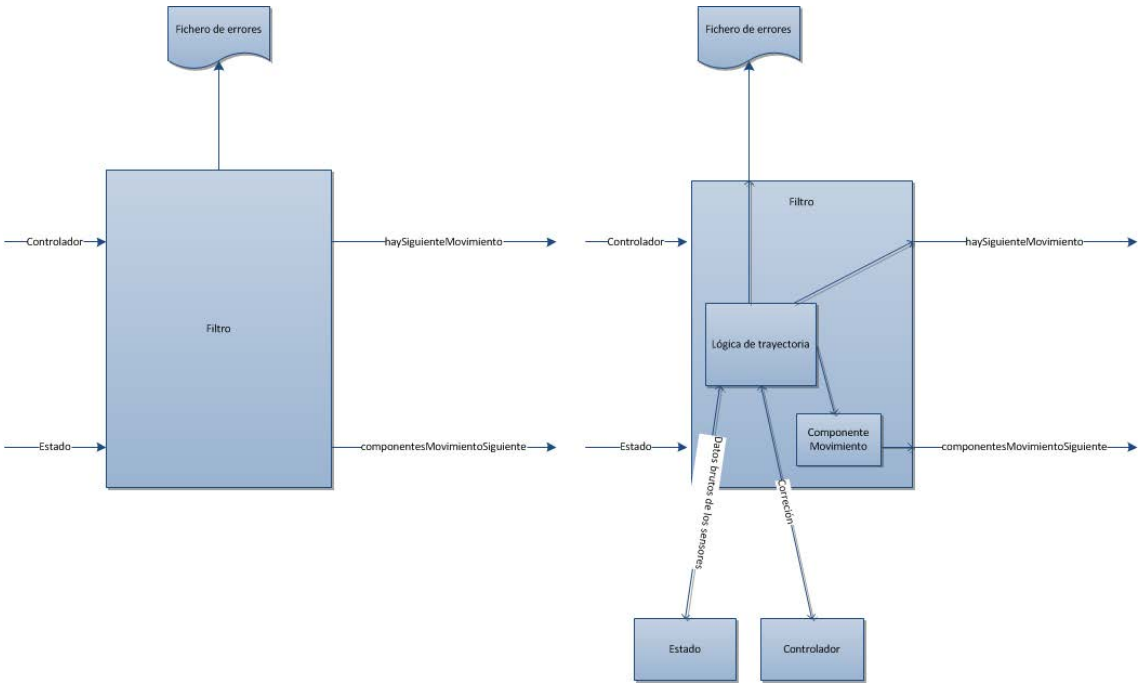


Figura 19 - Diagrama de funcionamiento del componente *Filtro*

La funcionalidad principal del *Filtro* deberá ser la siguiente:

- Conocer e implementar el objetivo del programa. Esto significa que es este componente quién conoce cuales son las metas del movimiento del robot, los objetivos y el modo en que se alcanzan. Por ejemplo, si el sistema debe pasar por una serie de trayectorias este componente deberá ser aquel que lea los puntos por los que deberá pasar, establecerá cuando se considera que el robot ha pasado por un punto y si ha terminado o no toda la trayectoria deseada.
- Tiene como objetivo alcanzar las metas que en el se desarrollan y para ello requiere de un controlador. El *Filtro* es el único que interactúa con el controlador para realizar los cálculos necesarios a la hora de proveer al exterior el siguiente movimiento a realizar.
- Deberá ofrecer una interfaz con los movimientos a realizar para que sea el bucle principal quien realice el *Movimiento* y no el *Filtro*. Esto se decidió así porque en la definición del *Filtro* solo se estableció que este actuaría como cálculo del movimiento pero no como actuador del movimiento en sí. Esta decisión ayuda a la escalabilidad y extensibilidad del sistema añadiendo la posibilidad de que varios filtros sean los que tomen decisiones conjuntas o simplemente para conocer el cálculo del movimiento pero no hacerlo y anotar las posibles variaciones del sistema.
- El *Filtro* escribirá en un fichero los errores que se quieran mostrar para realizar pruebas.
- El *Filtro* deberá leer los datos iniciales de los controladores, trayectorias, puntos y cualquier otro dato que se deba establecer antes de la ejecución.

Estado

El *Estado* es el componente que proporciona una interfaz con los sensores y dispositivos del robot.

Figura 20 - Diagrama de funcionamiento del componente *Estado*

Las funciones de este componente son sencillas:

- Ofrecer una interfaz con los datos de los sensores par que el sistema conozca cual es el estado actual del robot.
- Ofrecer una interfaz de comunicación con los otros componentes para ejecutar los movimientos que estos le digan.
- En el caso de se requerido, dibujar la trayectoria del robot a lo largo del tiempo.

El componente *Filtro* y el componente *Estado* son dos componentes altamente conexos ya que los datos que proporciona el *Estado* son muy diferentes dependiendo de si este es simulado, real, con dispositivos de cámara únicamente, o dispositivos de otra índole. Por ello, el filtro deberá conocer qué *Estado* está utilizando para obtener los datos pertinentes en cada situación.

Controlador

El componente *Controlador* tiene como único objetivo, dado un valor objetivo y un valor actual, el valor requerido para llegar al objetivo. Tal y como dice la definición de controlador este componente es indiferente del sistema que se está utilizando y, por tanto debe ofrecer una interfaz simple y no compleja que pueda utilizarse con cualquier sistema sin depender de la arquitectura simulada o real.

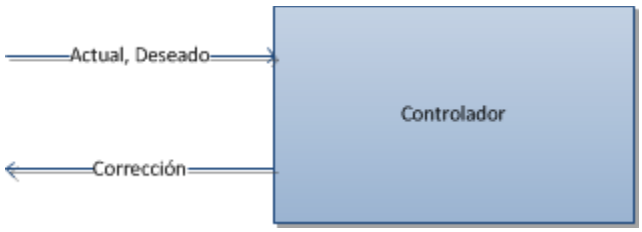


Figura 21 - Diagrama de funcionamiento del componente *Controlador*

Movimiento

El componente *Movimiento* es aquel que inicializa todos los demás componentes, alberga el bucle principal y ejecuta los movimientos que el *Filtro* le predice.

Para terminar con esta sección y como nexo con la implementación se puede observar el diagrama de secuencia que sigue la ejecución del software y sirve como apoyo a futuras implementaciones sobre el mismo.

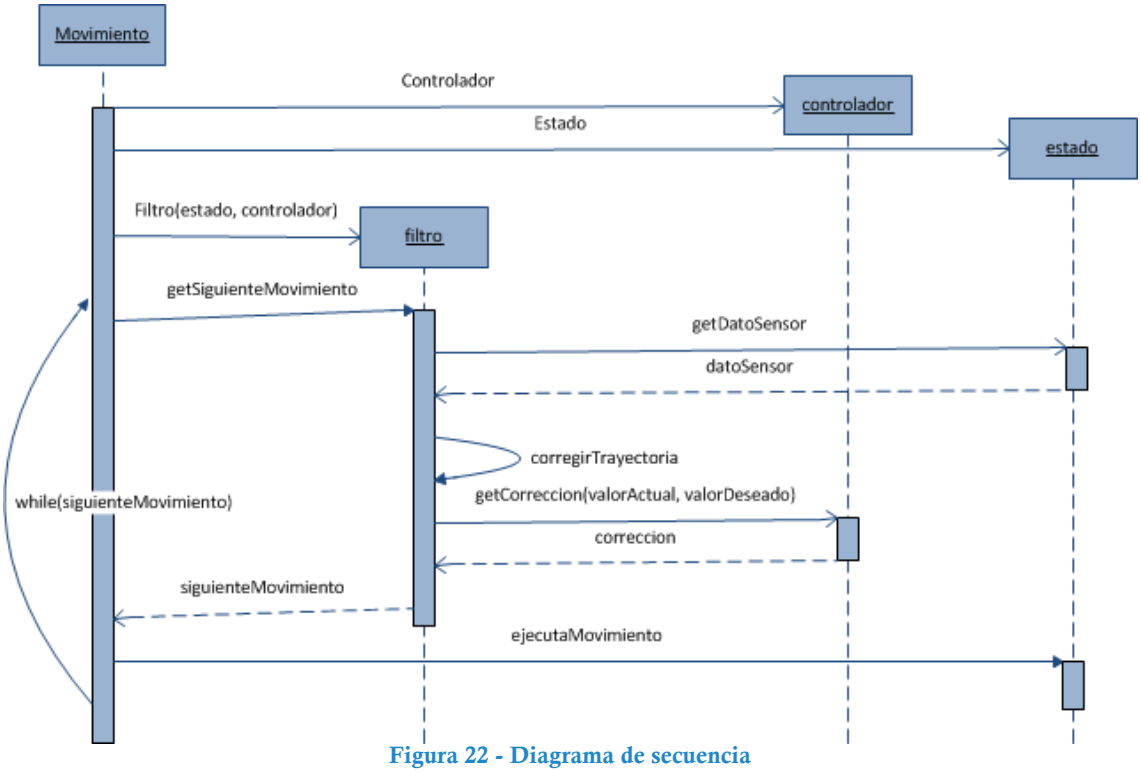


Figura 22 - Diagrama de secuencia

El movimiento, tras instanciar el *Controlador* y el *Filtro*, entra en el bucle donde, mientras queden movimientos por realizar, estará permanentemente. En ese bucle interacciona con *Filtro* pidiéndole el siguiente movimiento a realizar. *Filtro* obtiene los datos del componente *Estado* en bruto, los adapta al sistema que el determina que deberá usarse para corregir la trayectoria. Esta funcionalidad es la que da nombre al componente *Filtro* ya que filtra los datos obtenidos por el *Estado* y los prepara para ejecutar el movimiento. Más tarde los envía al componente *Controlador* que le devuelve el dato corregido y actualizado por el mismo. En el caso de un controlador PID este devolverá el dato corregido. Esta información será el movimiento que se deberá realizar posteriormente. Una vez devuelto el valor de siguiente movimiento (si existe o no otro movimiento) el componente *Movimiento* ejecuta el dato de movimiento que ha calculado *Filtro.*

Implementación del software.

Durante esta sección se explicará la implementación del software resaltando los aspectos más importantes de esta. El contenido tendrá la siguiente estructura: primero se describirá el modelo general del software y luego se dividirá en dos subsecciones. Estas dos subsecciones son las correspondientes a la implementación simulada y a la implementación con dispositivos reales, en este caso, la cámara.

La fase de implementación responde de una manera fiel al diseño realizado previamente utilizando un lenguaje de programación apropiado y necesario para este proyecto. El lenguaje en cuestión es C++. Ha sido elegido por ser el lenguaje con el que se puede comunicar con *Player/Stage* y con los proyectos en los que sustentan las bases de este (Rebollo, 2010).

Durante las páginas de este apartado se describirá el modelo más general del sistema y su versión específica ya sea simulada o con dispositivos reales. Se tratará de incluir el menor código posible dado que este ya existe en el código adjunto a este proyecto, pero en ocasiones puede que visualizar una parte de ese código sea necesario para comprender el funcionamiento de la aplicación.

Diagrama de clases

La siguiente figura representa el diagrama de clases de la aplicación en una versión reducida sin incluir los métodos de cada clase para hacerlo más legible.

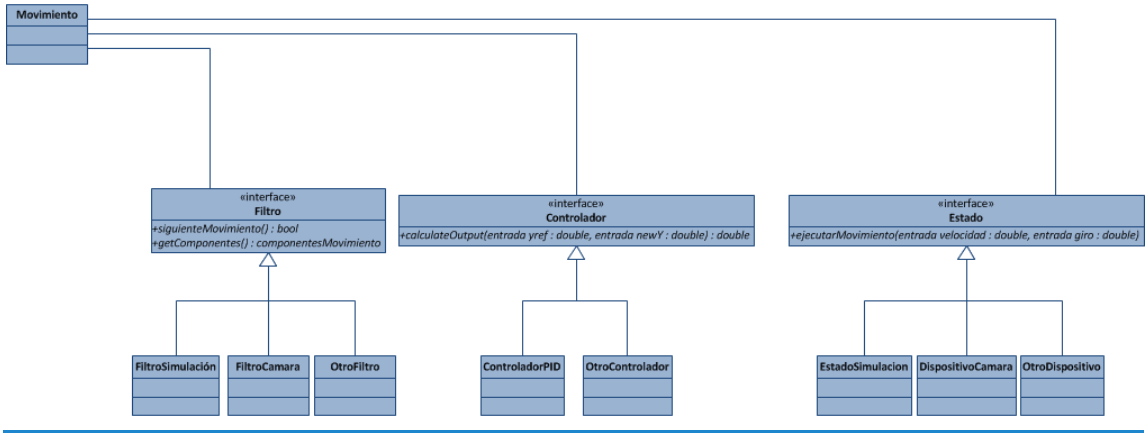


Figura 23 - Diagrama de clases

Los llamados componentes anteriores (*Filtro, Controlador* y *Estado*) son interfaces de las que heredan las implementaciones reales tales como *FiltroSimulacion*, *ControladorPID* o *EstadoSimulacion.*

Implementación detallada: Desarrollo Simulado.

En este apartado se explicará la implementación del proyecto en el entorno simulado.

Clases del componente Filtro

La clase *Filtro* tiene como objetivo principal implementar la lógica de movimiento y trayectoria del robot, generar los ficheros de error y comunicarse con el estado del robot y el controlador.

A continuación se muestra el diagrama y la descripción del componente *Filtro* y *FiltroSimulacion.*

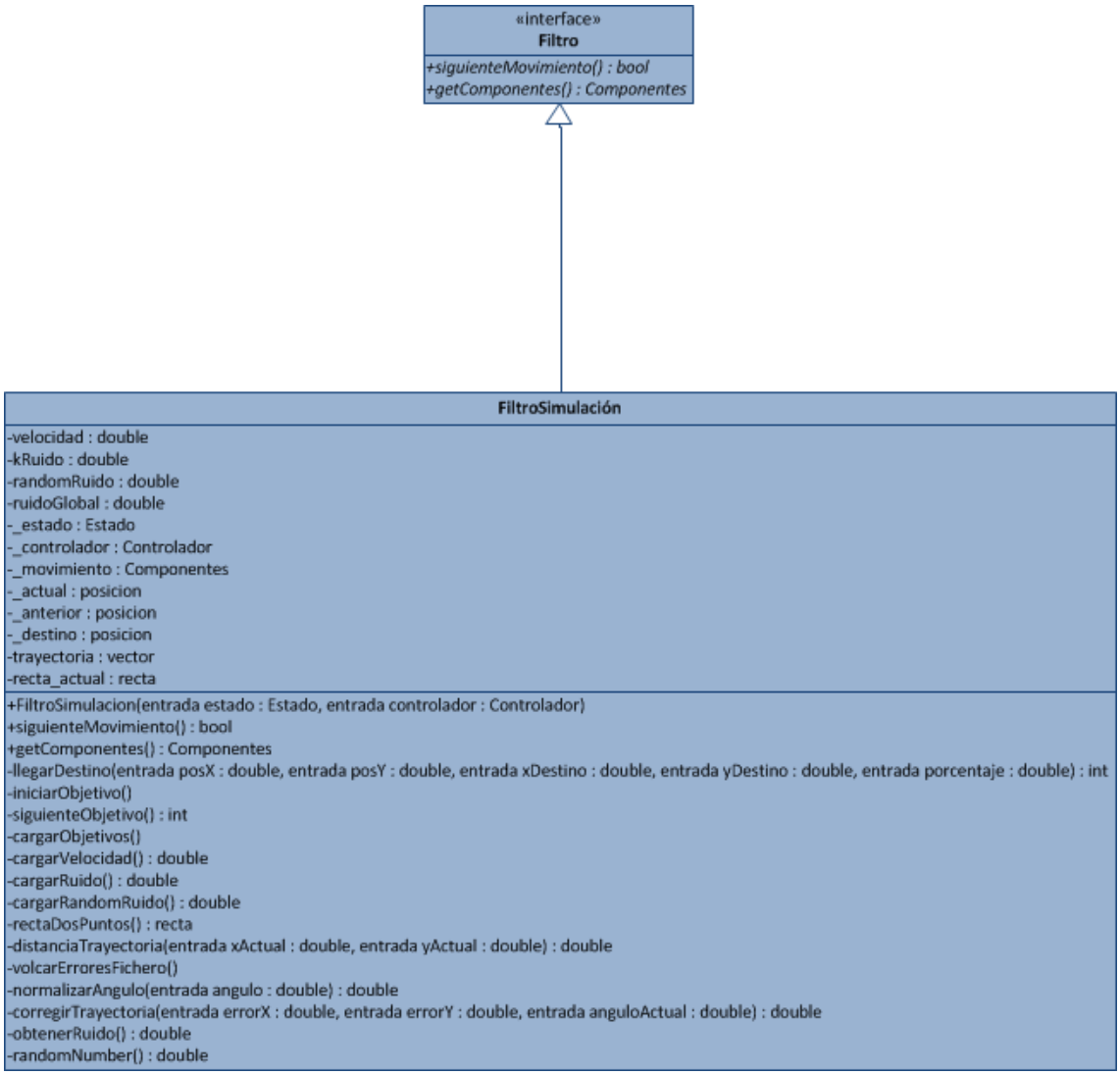


Figura 24 - Diagrama de las clases *Filtro* y *FiltroSimulacion*

A continuación se detallarán cada uno de los atributos y métodos.

Atributos FiltroSimulacion

Nombre Atributo	Tipo Atributo	Descripción Atributo
_estado	EstadoSimulacion	Objeto de tipo Estado que proporciona y modifica el estado del robot.
_controlador	Controlador	Objeto de tipo controlador que proporciona el o los métodos para obtener una corrección dada una entrada ideal.
_movimiento	*(struct)* componentesMovimiento	Componentes del movimiento a realizar en la siguiente iteración del bucle. Esta estructura contiene un *double* giro y otro de velocidad.
_velocidad	double	Velocidad actual del robot.
_actual	*(struct)* posicion	La posición actual en la que se encuentra el robot. Pese a que el struct contiene más variables, las que suelen usarse son *posicionX* y *posicionY*
_destino	*(struct)* posicion	La posición a la que se desea llegar con el robot.
_anterior	*(struct)* posición	La última posición conocida del robot. Sirve para dibujar línea en el simulador.
_kRuido	double	Sesgo del ruido que se utilizará en la simulación
_randomRuido	double	Variable que define si el ruido es global o aleatorio
_ruidoGlobal	double	Ruido Global generado al aleatoriamente una vez si se desea generar un ruido aleatorio global o una vez cada iteración del bucle si se desea crear aleatoriamente.

_trayectoria	vector<posicion>	Trayectoria por la que deberá pasar el robot para concluir la ejecución
_iter	vector<posicion>::iterator	Iterador del anterior vector.
_errores	vector<double>	Vector de errores que se acumulan en el recorrido de la trayectoria.
_rectaActual	*(struct)* recta	Contiene la recta que va desde el anterior punto al punto actual.

Métodos FiltroSimulacion

Nombre del método	Visibilidad	Descripción
siguienteMovimiento	public	Devuelve un bool indicando si existe un movimiento siguiente o no. Además, en el calcula dicho movimiento y lo almacena en la variable _movimiento.
getComponentes	public	Devuelve el atributo _movimiento.
llegarDestino	private	Devuelve si se ha llegado al destino o no dado un sesgo, la posición de destino y la actual.
inicializarObjetivo	private	Función que inicializa la variable _destino con el primer punto de la trayectoria.
siguienteObjetivo	private	Establece el objetivo siguiente de la lista de trayectoria.
cargarObjetivos	private	Método que lee del fichero de trayectoria lo vuelca sobre el vector _trayectoria.
cargarVelocidad	private	Método que lee del fichero de opciones en busca de la velocidad lo vuelca sobre la variable velocidad.

cargarKRuido	private	Método que lee del fichero de opciones en busca del ruido y lo vuelca sobre la variable kRuido.
cargarRandomRuido	private	Método que lee del fichero de opciones en busca del bool RandomRuido y lo vuelca sobre la variable _randomRuido.
rectaDosPuntos	private	Devuelve un struct con la recta que va desde el punto anterior al actual.
distanciaTrayectoria	private	Devuelve la distancia de la recta trayectoria del punto actual.
volcarErroresFichero	private	Almacena el vector de errores en el fichero de nombre homónimo.
normalizarAngulo	private	Función que, dado un ángulo, devuelve el giro menos costoso de realizar para recrearlo.
corregirTrayectoria	private	Función que corrige la trayectoria mediante el controlador
obtenerRuido	private	Función que devuelve el ruido.
randomNumber	private	Devuelve el ruido generado aleatoriamente dependiendo de si es global o aleatorio a cada iteración del bucle.

Dado que la breve explicación de la tabla puede resultar insuficiente, a continuación se incluye una pequeña explicación en pseudocódigo y comentarios del método *siguienteMovimiento*().

Preparaciones previas a la lógica de programa.

```
error = _destino - posicionActual + ruido
_errores.AñadirElError( distanciaTrayectoria(posicionActual))
nextMov = corregirTrayectoria(error, _estado.Orientacion)
_actual = posicionActual + ruido
recta_actual = rectaDosPuntos()
```

Lógica del método.

```
Si llegarDestino(0.5) = true Entonces //#1
      _anterior = _destino
      recta_actual = rectaDosPuntos()
      Si siguienteObjetivo() = 0 Entonces
      //Se ha llegado al punto. Fin de la trayectoria
            _errores.Añadir(muesca)
            _movimiento = nulo
            volcarErroresFichero()
            continua = false
      SiNo Entonces
      //Se ha llegado al punto. Siguiente punto
            _errores.push_back(siguientePunto)
            _movimiento = nextMov
            continua = true
      Fin //siguienteObjetivo
SiNo Entonces
      velocidadReducida = velocidad
      Si llegarDestino(2) Entonces //#2
      //Se acerca al punto. Se decrementa la velocidad
            velocidadReducida = velocidad / 1.5
      Fin //llegarDestino #2
      _movimiento = nextMov
      continua = true
Fin //llegarDestino #1
```

Parámetro de retorno.

```
devolver continua
```

A continuación se muestra el diagrama y la descripción del componente *Filtro* y *FiltroCamara.*

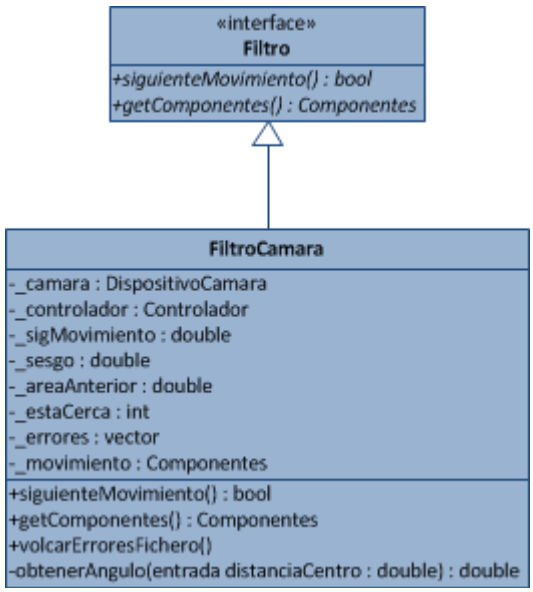


Figura 25 - Diagrama de las clases *Filtro* y *FiltroCamara*

Atributos FiltroCamara

Nombre Atributo	Tipo Atributo	Descripción Atributo
_camara	DispositivoCamara	Objeto de tipo Estado que proporciona los datos del dispositivo real cámara y modifica el estado del robot en el entorno simulado.
_controlador	Controlador	Objeto de tipo controlador que proporciona el o los métodos para obtener una corrección dada una entrada ideal.
_movimiento	*(struct)* componentesMovimiento	Componentes del movimiento a realizar en la siguiente iteración del bucle. Esta estructura contiene un *double* giro y otro de velocidad.
_velocidad	double	Velocidad actual del robot.
_errores	vector<double>	Vector de errores que se acumulan en el recorrido de la trayectoria.

_sigMovimiento	double	Resultado obtenido por el controlador PID tras prever el siguiente movimiento basándose en los datos de la cámara.
_sesgo	double	Variable que sirve como sesgo al giro a realizar por el robot tras obtener el ángulo de giro.
_areaAnterior	double	Valor del área antes de una nueva iteración del bucle.
_estaCerca	int	Variable auxiliar del sistema que delimita si el sistema está cerca o no. Dependiendo de su valor se añadirá una nueva tupla a los errores o no.

Métodos FiltroCamara

Nombre del método	Visibilidad	Descripción
siguienteMovimiento	public	Devuelve un bool indicando si existe un movimiento siguiente o no. Además, en el calcula dicho movimiento y lo almacena en la variable _movimiento. En este caso haciendo uso de los datos obtenidos de la cámara.
getComponentes	public	Devuelve el atributo _movimiento.
volcarErroresFichero	private	Almacena el vector de errores en el fichero de nombre homónimo.
obtenerAngulo	double	Función que, dado la distancia al centro de la imagen del objeto detectado, devuelve el ángulo de giro a realizar para posicionar la cámara en el centro del objeto.

A continuación se describirá el método *siguienteMovimiento* de manera más específica para facilitar la comprensión del mismo.

```
desvio  = _camara.obtenerDesvioOrientacion
Si    (AREAMAXIMA   >=   _camara.obtenerArea)   Y   (SeEncuentraObjeto)
Entonces//#1
      _sigMovimiento = _controlador.calculateOutput(0.0,desvio)
      _movimiento.velocidad = 0.7
      _movimiento.giro = obtenerAngulo(_sigMovimiento)
      _areaAnterior = camara.obtenerArea
SiNo
      _camara.obtenerDesvioOrientacion
      _movimiento.velocidad = 0
      _movimiento.giro = 0
Fin //#1
```

El sistema debe seguir al objeto detectado y detenerse cuando se encuentra a cierta distancia de él.

Clases del componente Estado

La clase tiene como objetivo principal ofrecer al resto de clases datos del robot tales como las coordenadas, la dirección de giro, batería… Además, este componente es el encargado de realizar los movimientos del robot. Por último, el componente *Estado* será el encargado de dibujar la trayectoria realizada en la simulación (en el caso de que la ejecución sea simulada).

A continuación se muestra el diagrama de las clases *Estado* y *EstadoSimulacion.*

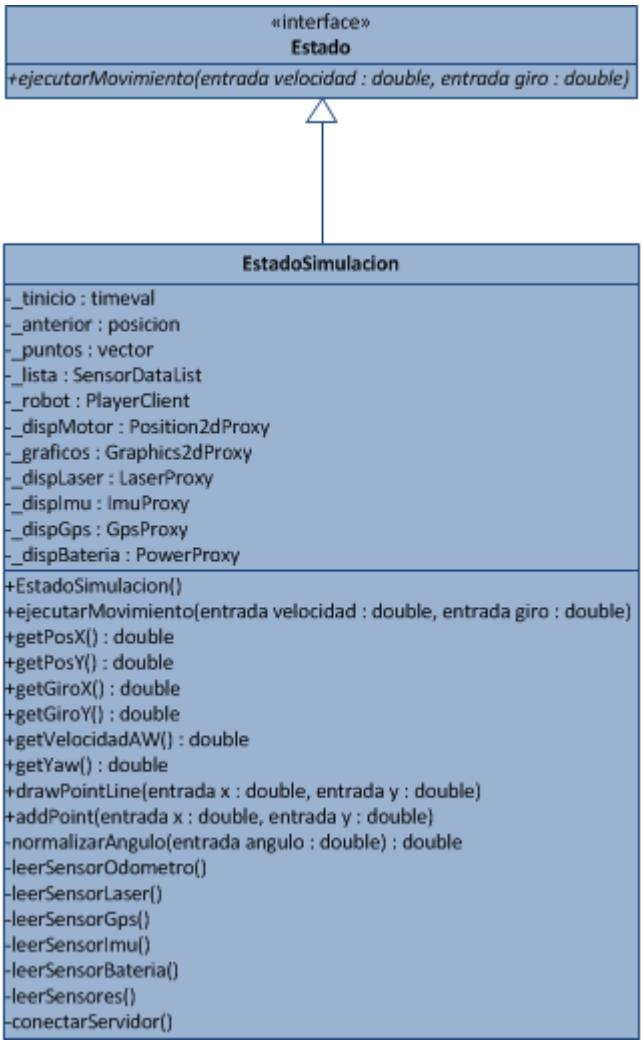


Figura 26 - Diagrama de las clases *Estado* y *EstadoSimulacion*

A continuación se detallarán cada uno de los atributos y métodos.

Atributos de la clase EstadoSimulacion

Nombre Atributo	Tipo Atributo	Descripción Atributo
_tinicio	*(struct)* timeval	Este atributo es necesario para establecer un temporizador en los dispositivos que serán leidos.
_anterior	*(struct)* posicion	Posición anterior del robot para poder dibujar en el simulador la trayectoria.
_puntos	*(struct)* punto	Puntos de trayectoria para poder dibujarlos en el simulador
_lista	SensorDataList	Lista de sensores inicializados
_robot	PlayerClient	Cliente de Player para realizar los movimientos al robot y obtener las posiciones
_dispMotor	Position2dProxy	Proxy que simula la posición del motor y su orientación
_graficos	Graphics2dProxy	Proxy que sirve para dibujar en el simulador la trayectoria ideal y la realizada.
_dispLaser	LaserProxy	Proxy que simula el laser
_dispImu	ImuProxy	Proxy que simula el dispositivo Imu
_dispGps	GpsProxy	Proxy que simula el dispositivo GPS para obtener su posición global.
_dispBateria	PowerProxy	Proxy que simula el dispositivo de batería

Métodos de la clase EstadoSimulacion

Nombre del método	Visibilidad	Descripción
EstadoSimulacion	public	Constructor de la clases EstadoSimulación
ejecutarMovimiento	public	Método que, mediante la comunicación con los dispositivos, ejecuta el movimiento deseado
getPosX	public	Devuelve la posición X del robot
getPosY	public	Devuelve la posición Y del robot
getGiroX	public	Devuelve la velocidad de giro en el eje X
getGiroY	public	Devuelve la velocidad de giro en el eje Y
getVelocidadAW	public	Devuelve la velocidad angular de giro.
drawPointLine	public	Dibuja una línea entre el punto anterior y el pasado por parámetro
addPoint	public	Añade un punto a la trayectoria a dibujar
normalizarAngulo	private	Función que, dado un ángulo, devuelve el giro menos costoso de realizar para recrearlo.
leerSensorOdometro	private	Devuelve los datos del sensor de odometría
leerSensorLaser	private	Devuelve los datos del sensor de Laser
leerSensorGps	private	Devuelve los datos del sensor GPS
leerSensorImu	private	Devuelve los datos del sensor de Imu
leerSensorBateria	private	Devuelve los datos del sensor de la batería
leerSensores	private	Función que llama a todas las funciones de lectura de sensores para actualizar sus datos.
conectarServidor	private	Función que conecta con el simulador de Player.

A continuación se muestra el diagrama de las clases *Estado* y *DispositivoCamara.*

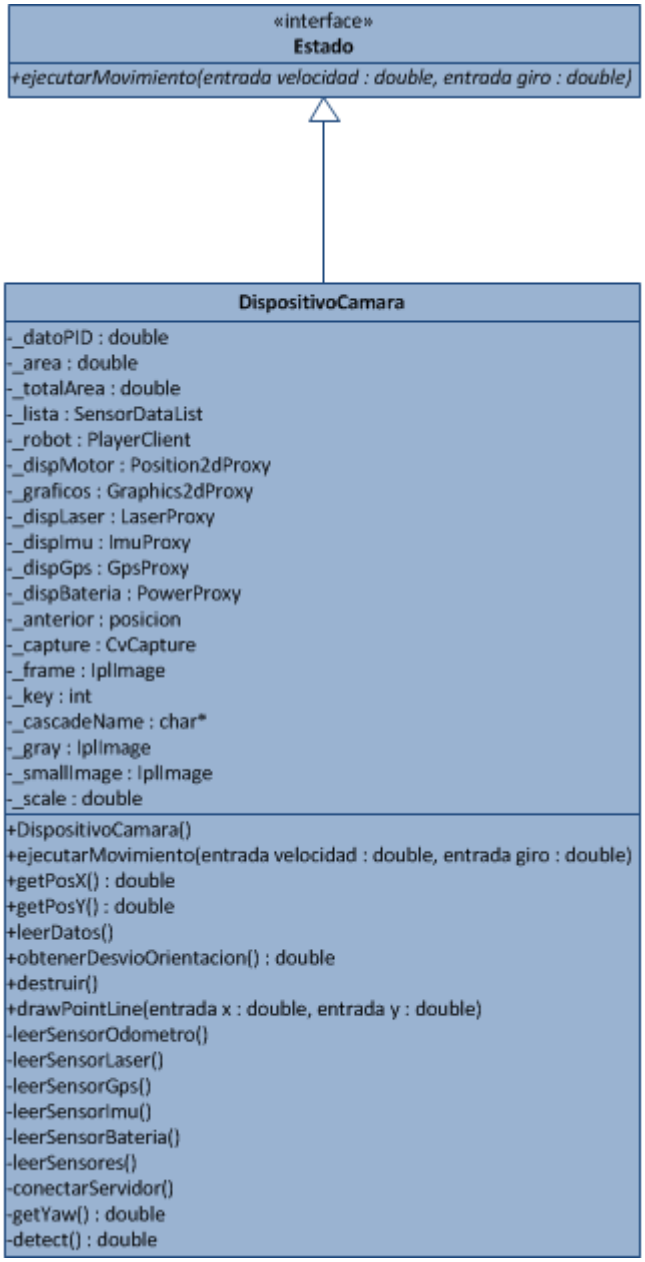


Figura 27 - Diagrama de las clases *Estado* y *DispositivoCamara*

A continuación se detallarán cada uno de los atributos y métodos.

Atributos de la clase DispositivoCamara

Nombre Atributo	Tipo Atributo	Descripción Atributo
_datoPID	double	Dato que se devolverá para el PID que mide la distancia desde el centro de la imagen al centro de la cara reconocida.
_area	double	Área del rectángulo de reconocimiento de caras.
_lista	SensorDataList	Lista de sensores inicializados
_robot	PlayerClient	Cliente de Player para realizar los movimientos al robot y obtener las posiciones

_dispMotor	Position2dProxy	Proxy que simula la posición del motor y su orientación
_graficos	Graphics2dProxy	Proxy que sirve para dibujar en el simulador la trayectoria ideal y la realizada.
_dispLaser	LaserProxy	Proxy que simula el laser
_dispImu	ImuProxy	Proxy que simula el dispositivo Imu
_dispGps	GpsProxy	Proxy que simula el dispositivo GPS para obtener su posición global.
_dispBateria	PowerProxy	Proxy que simula el dispositivo de batería
_anterior	*(struct)* posicion	Posición anterior del robot para poder dibujar en el simulador la trayectoria.
_capture	CvCapture	Capturador de imágenes. En el dispositivo de la cámara es del sistema.
_frame	IplFrame	Frame capturado por la cámara
_gray	IplFrame	Almacena el atributo _frame en escala de grises.
_smallImage	IplFrame	Almacena el atributo _frame con una escala de tamaño menor.
_scale	double	Escala que sirve como sesgo al sistema para devolver las medidas de la imagen.

Métodos de la clase DispositivoCamara

Nombre del método	Visibilidad	Descripción
DispositivoCamara	public	Constructor de la clases EstadoSimulación
ejecutarMovimiento	public	Método que, mediante la comunicación con los dispositivos, ejecuta el movimiento deseado
getPosX	public	Devuelve la posición X del robot
getPosY	public	Devuelve la posición Y del robot
drawPointLine	public	Dibuja una línea entre el punto anterior y el pasado por parámetro
destruir	public	Destruye la conexión con el dispositivo de la cámara.

leerSensorOdometro	private	Devuelve los datos del sensor de odometría
leerSensorLaser	private	Devuelve los datos del sensor de Laser
leerSensorGps	private	Devuelve los datos del sensor GPS
leerSensorImu	private	Devuelve los datos del sensor de Imu
leerSensorBateria	private	Devuelve los datos del sensor de la batería
leerSensores	private	Función que llama a todas las funciones de lectura de sensores para actualizar sus datos.
conectarServidor	private	Función que conecta con el simulador de Player.
getYaw	private	Devuelve la velocidad angular de giro.
detect	private	Detecta las caras en la imagen o atributo _frame.

Debido a que la breve explicación de la función *detect* puede resultar confusa se va a incluir un listado de acciones por las que pasa para detectar la cara en la imagen:

1. Comenzando en la función *obtenerDesvío*, se obtiene un frame con la captura de la cámara.
2. Este frame se pasa a escala de grises
3. Se guarda una versión del frame a tamaño reducido.
4. Se llama a la función *detect*.
    a. Mediante la una función detectora de objetos *Haar* se obtiene el número de caras. Esta función toma como entrenamiento el fichero incluido en el código *haarcascade_frontalface_alt2.xml* .
    b. Se obtiene el dato de distancia entre el centro de la imagen y el centro del rectángulo donde se ha encontrado la cara. Se almacena en una variable auxiliar llamada *dato_PID*.
    c. Se calcula el área del rectángulo donde se ha encontrado la cámara y se iguala a la variable *_area*.
    d. Se devuelve la variable *dato_PID*.
5. Se devuelve la variable *_datoPID* resultado de la función detect.

El autor de este documento ha decidido obviar las instrucciones propias de las librerías de *OpenCV* por considerarlas demasiado técnicas para ser relevantes en la lectura de esta sección.

Clases del componente Controlador

El componente *Controlador* tiene como objetivo principal, dado un dato de estado real y un dato de objetivo, obtener la aproximación necesaria a ese dato teniendo en cuenta errores de medida, posibles sesgos… Este componente debe ser totalmente ajeno al sistema y con muy poco nivel de dependencia con el mismo

A continuación se muestra el diagrama de las clases *Controlador* y *ControladorPID.*

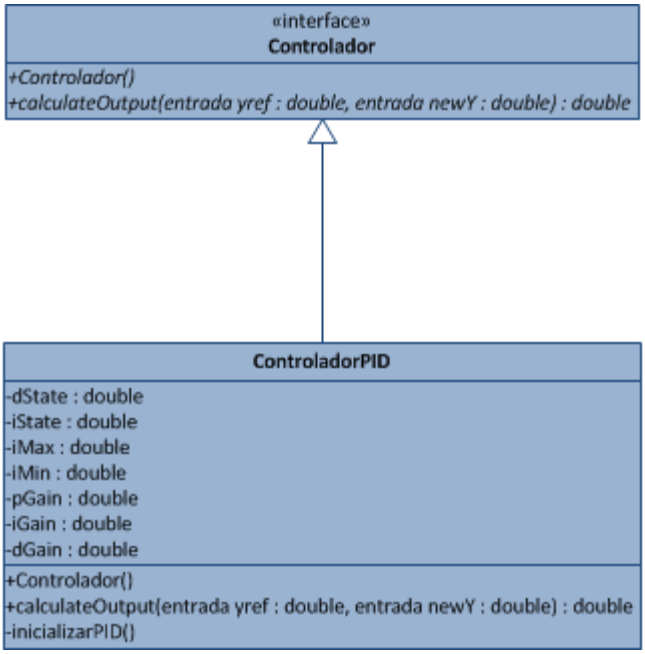


Figura 28 - Diagrama de las clases *Controlador* y *ControladorPID*

A continuación se detallarán cada uno de los atributos y métodos.

Atributos de la clase ControladorPID

Nombre Atributo	Tipo Atributo	Descripción Atributo
_dState	double	Atributo que almacena el estado de la variable *d.*
_iState	double	Atributo que almacena el estado de la variable *i.*
_iMax	double	Sesgo en el máximo de la variable *i* para que esta no cree un controlador inútil.
_iMin	double	Sesgo en el mínimo de la variable *i* para que esta no cree un controlador inútil.
_pGain	double	Variable *Kp* que equivale a la ganancia por la que es multiplicada el valor *p* obtenido.

_iGain	double	Variable *Ki* que equivale a la ganancia por la que es multiplicada el valor *i* obtenido.
_dGain	double	Variable *Kd* que equivale a la ganancia por la que es multiplicada el valor *d* obtenido.

Métodos EstadoSimulacion

Nombre del método	Visibilidad	Descripción
Controlador	public	Método constructor de la clase *controladorPID*. Este constructor inicializa las ganancias leyéndolas del fichero *ficheroPID.*
calculateOutput	public	Devuelve la estimación necesaria para llegar al objetivo pasado por parámetro basándose en los parámetros Proporcional, Integral y Derivativo.
inicializarPID	private	Método que es llamado por le constructor para inicializar el controlador PID con los parámetros del fichero *ficheroPID.*

El método *calculateOutput* sigue el siguiente pseudocódigo (anteriormente mostrado en la sección de controladores, estado del arte).

```
error = entradaDeseada - entradaActual
TérminoP = GananciaP * error.
EstadoI = EstadoI + error
Si (EstadoI > EstadoIMáximo) Entonces EstadoI = EstadoIMáximo
Si (EstadoI < EstadoIMínimo) Entonces EstadoI = EstadoIMínimo
TérminoI = GananciaI * EstadoI
TérminoD = GananciaD * (error - EstadoD)
EstadoD = error
Resultado = TérminoP + TérminoI – TérminoD.
```

Experimentación

Introducción y preparación de las pruebas.

Este apartado describe las pruebas que se han realizado al software desarrollado para este proyecto. Esta pruebas son esenciales tanto para calibrar el sistema como para comprender las posibles mejoras, trabajos sobre esta plataforma y futuras implementaciones de otros sistemas apoyados en esta capa software. En base a estas pruebas se realizan valoraciones y se sacarán algunas conclusiones que ayudaran a entender la motivación y comprensión de este proyecto.

Se dividirán las pruebas en dos apartados:

- Pruebas realizadas en el entorno virtual con *Player/Stage*
- Pruebas realizadas en el entorno virtual y el dispositivo óptico.

Los datos de error de este sistema han sido realizados mediante el siguiente procedimiento:

- Durante la ejecución se almacena en memoria los datos de error que se van calculando a cada iteración del algoritmo.
- Una vez terminado se vuelcan dichos resultados sobre un fichero.

Se ha decidido realizar las pruebas de esta manera para ahorrar tiempo de E/S al escribir en el fichero solo al final de cada prueba.

Esta implementación tiene una limitación clara y es que si el sistema no converge ni llega a sus puntos objetivos el fichero de error nunca se almacena y, por lo tanto, esa prueba (pese a que el no ser convergente ya es motivo suficiente de ineficiencia) no será valida.

Las pruebas realizadas en el entorno simulado están reflejadas mediante unos gráficos de líneas. Cada una de esas líneas representan la trayectoria seguida hasta llegar a un punto.

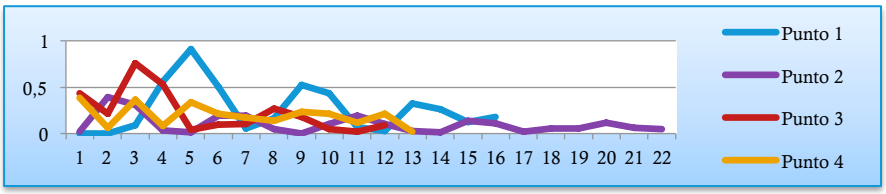


Figura 29 - Gráfico de ejemplo

El valor que toma el eje *y* de la gráfica es el error que existe entre a posición actual y la recta ideal. Es decir, la distancia del punto actual a la recta ideal.

Consideraciones en todas las pruebas:

1- Todas las pruebas han sido realizadas con los sesgos de *Ki* entre [-5,5]. Esto implica que esta variable intercederá solo aumentando o disminuyendo esa variable 5 puntos.
2- Durante el transcurso de este apartado se tomará como referencia temporal los pasos que necesita el sistema para llega a un punto. Esto viene reflejado en la gráfica en el eje x.

Cosas que contar en todos los casos

1- La velocidad es uno de los factores más importantes porque el intervalo de tiempo entre paso y paso del robot puede provocar que pierda el rumbo que debía mantener. Por lo tanto a mayor velocidad, menor precisión.

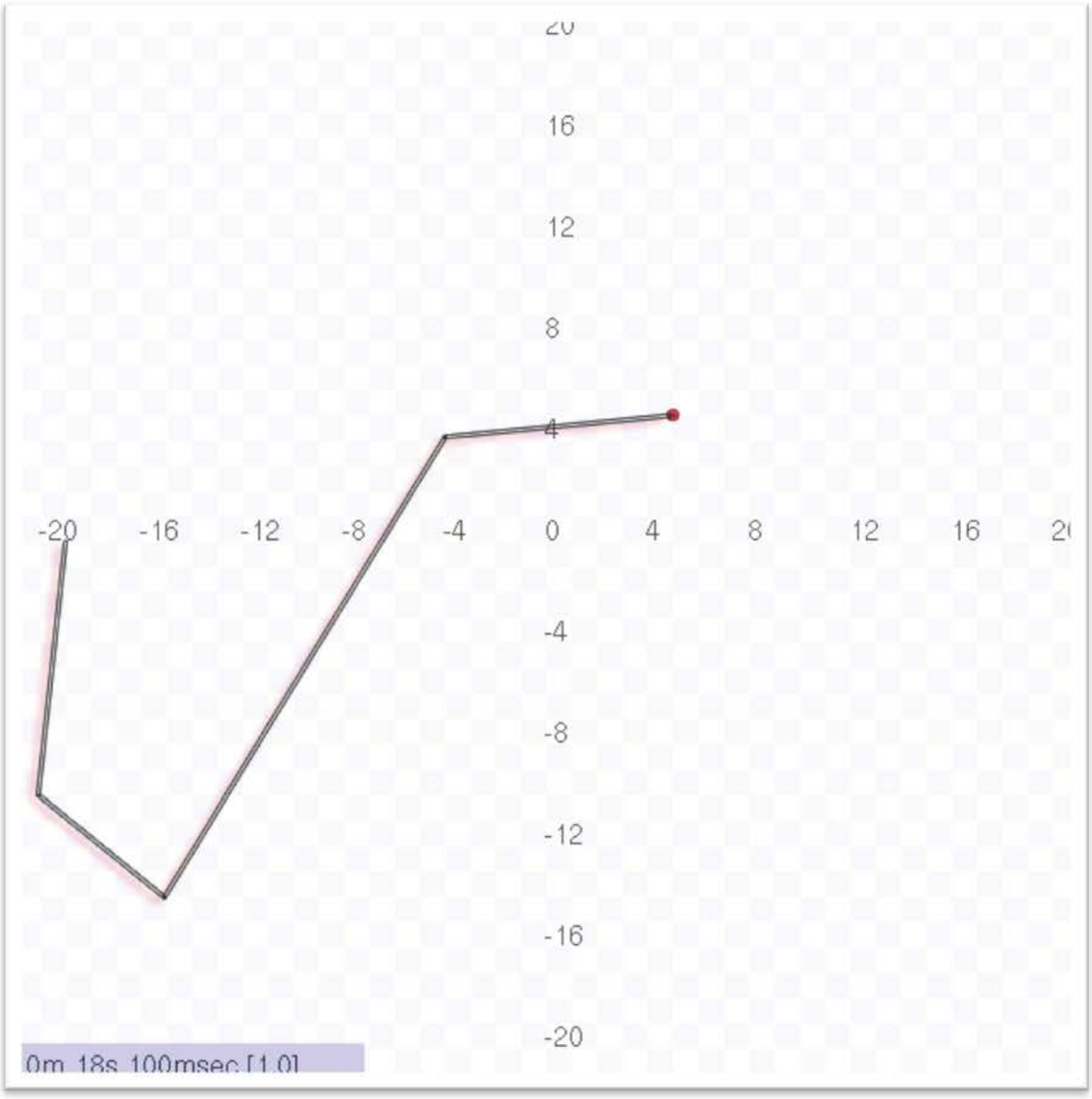
Pruebas iniciales.

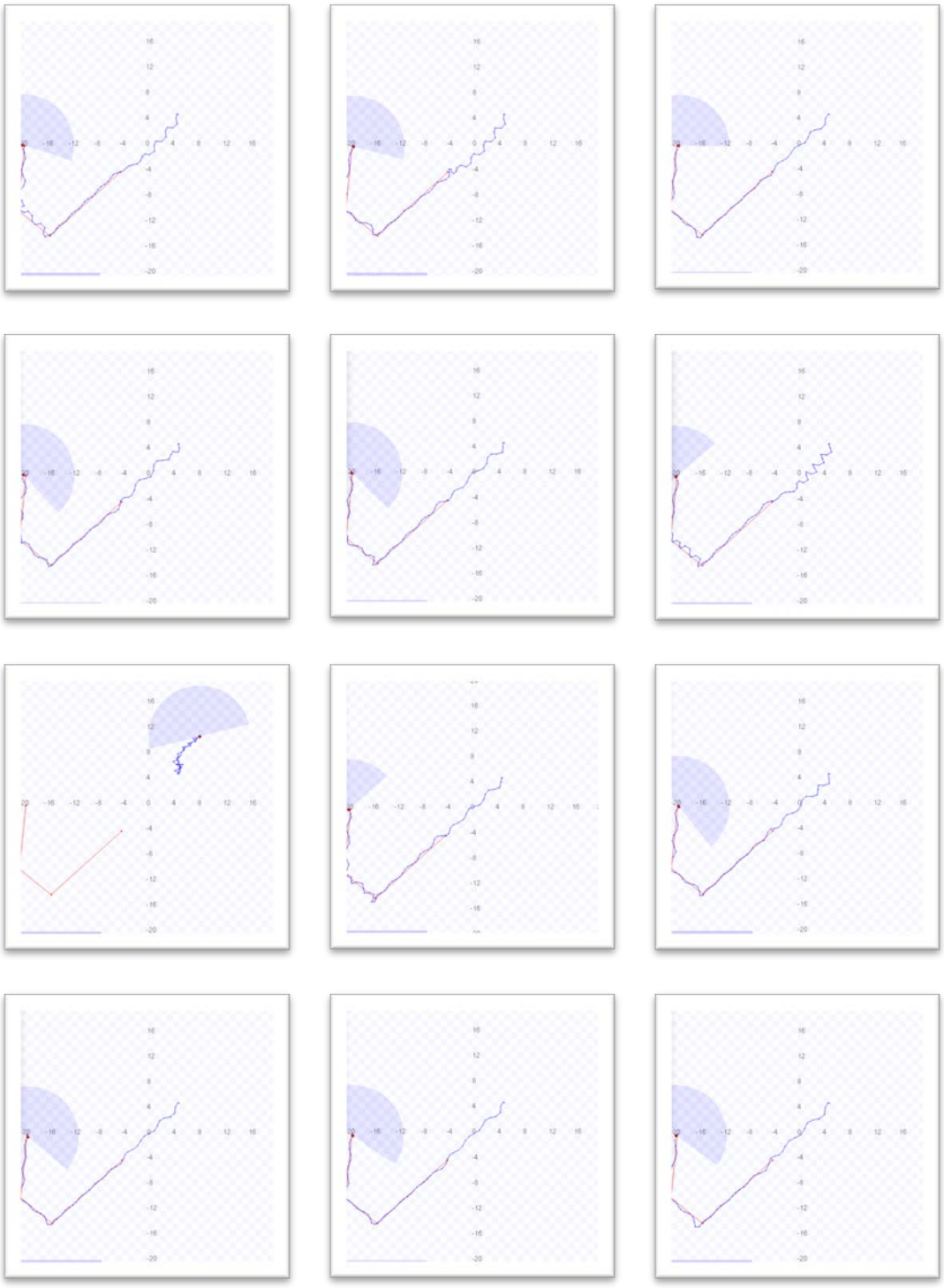
Estas pruebas iniciales han sido realizadas con el objetivo de probar el sistema y sesgar el intervalo de valores que pueden tomar los coeficientes del controlador para trabajar cómodamente sobre las pruebas relevantes.

Pruebas realizadas con la trayectoria: (-4,-4), (-15,-14), (-20,-10), (-19,0)

Prueba realizada	*V*	*Kp*	*Ki*	*Kd*
0	0,7	0,8	2	0
1	0,7	0,8	1,5	0
2	0,7	0,9	1,5	0
3	0,7	0,9	1,5	0,2
4	0,7	0,9	1,5	0,1
5	0,7	0,8	1,5	0,1
6	0,7	0,9	1,5	0,5
7	0,7	0,9	1,7	0,1
8	0,7	0,9	1,4	0,1
9	0,7	0,9	1,4	0
10	0,7	0,9	1	0
11	0,8	0,9	1,4	0

Esta prueba ha sido elegida con puntos al azar con el único objetivo de descartar valores en los coeficientes del controlador.

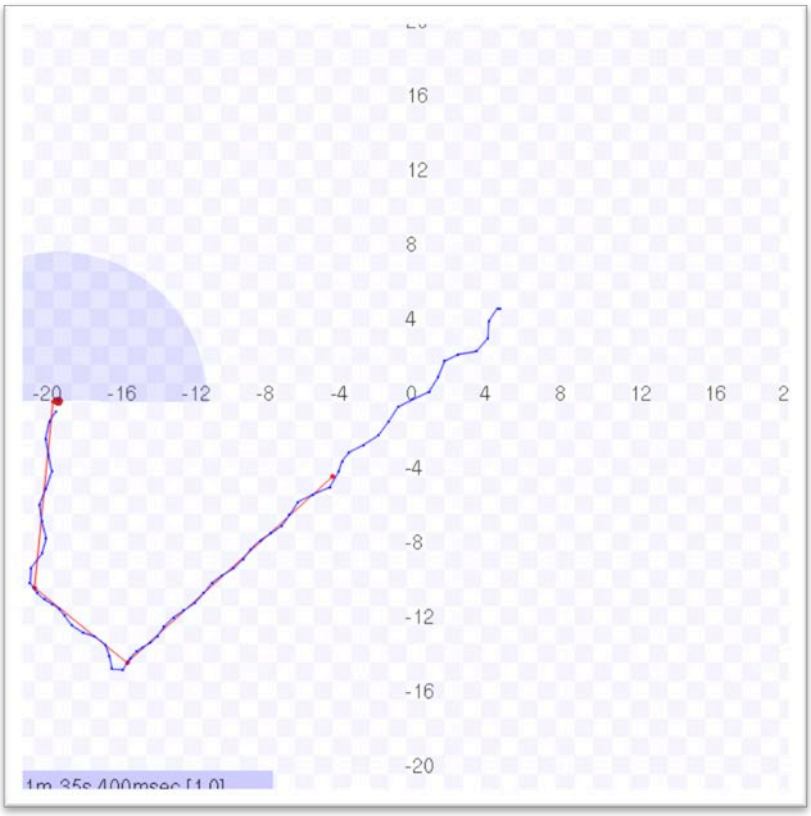


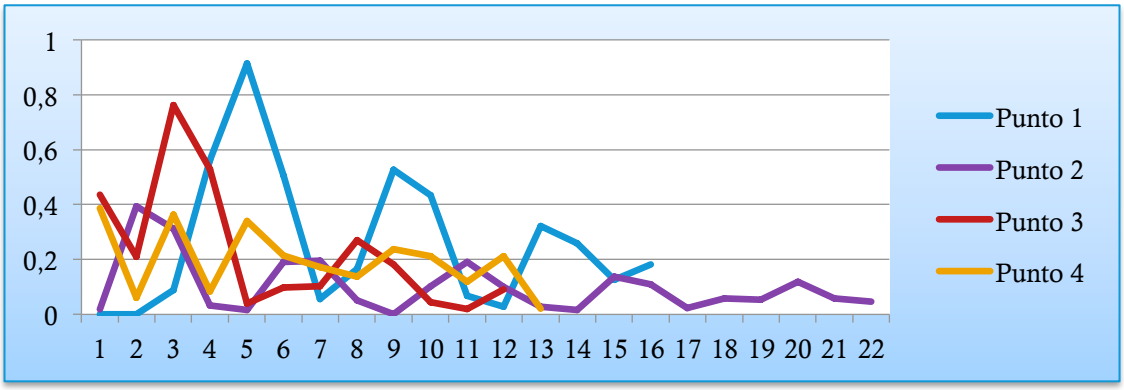


Se estudiarán en detalle aquellas pruebas que hayan resultado más interesantes para el proyecto. En este caso, las pruebas 2, 3,6, 10 y 11.

Prueba 2

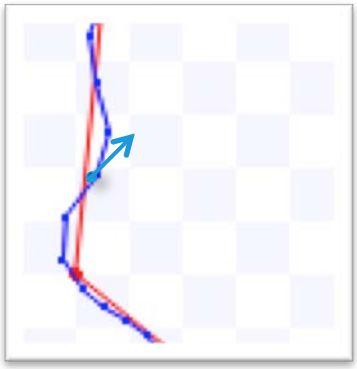
Esta prueba ha sido ejecutada con las variables: *Kp = 0.9, Ki = 1.5, Kd = 0* y una velocidad *V = 0.*





La imagen indica un pequeño periodo de adaptación al inicio de la trayectoria que se estabiliza cuando el robot va llegando al primer punto. Esto indica que el controlador necesita un periodo de adaptación antes de funcionar correctamente. La longitud de este periodo viene influenciada por la colocación inicial del robot, su dirección y sus sesgos (en este caso -5 y 5 como máximos y mínimos de *Ki*).

Si bien se puede comprobar un ligero alejamiento durante el principio del Punto 3, se puede contrastar con la imagen sosteniendo que es una alejamiento de dirección y no de distancia respecto a la trayectoria ideal.



Por lo tanto el controlador rectifica su dirección para colocarlo de nuevo en la trayectoria. Esto provoca un pico en la gráfica que llama la atención dado que el resto de la gráfica se mantiene más o menos estable.

Aun así, no se debe tomar esta variación muy en cuenta ya que retoma su trayectoria correctamente dando a lugar a un resultado bastante aceptable.

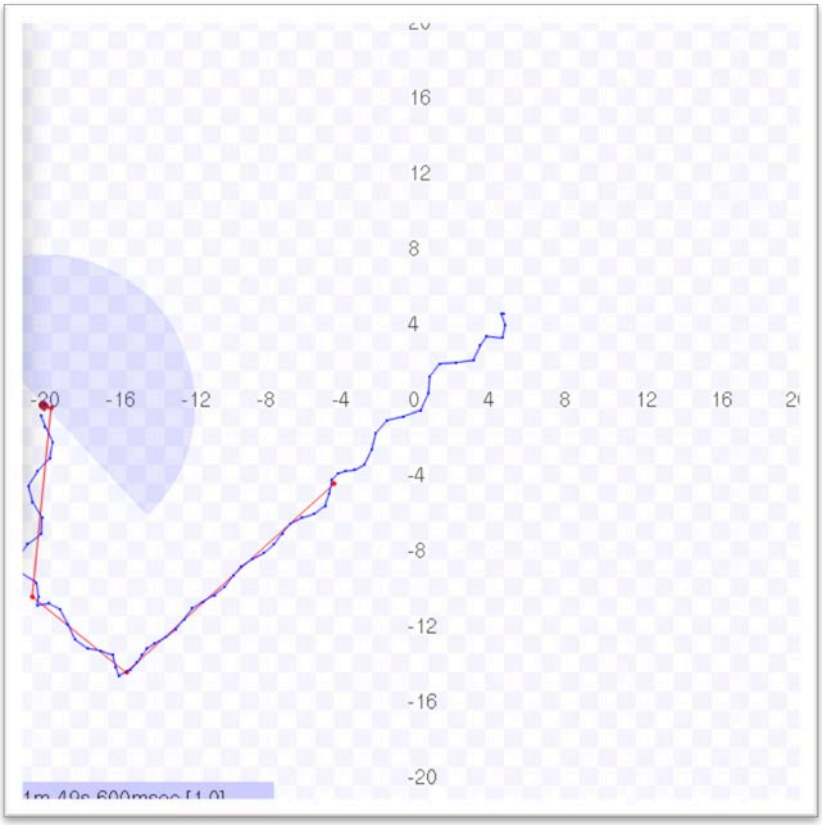
Si se observa con detenimiento el final del primer punto y entrada del segundo, se puede decretar que el sistema tiene problemas a la hora de realizar giros prolongados o cerrados (aquellos que sean de más de 30 grados) ya que necesita de varios pasos (temporales) para volver a mantenerse sobre la trayectoria ideal.

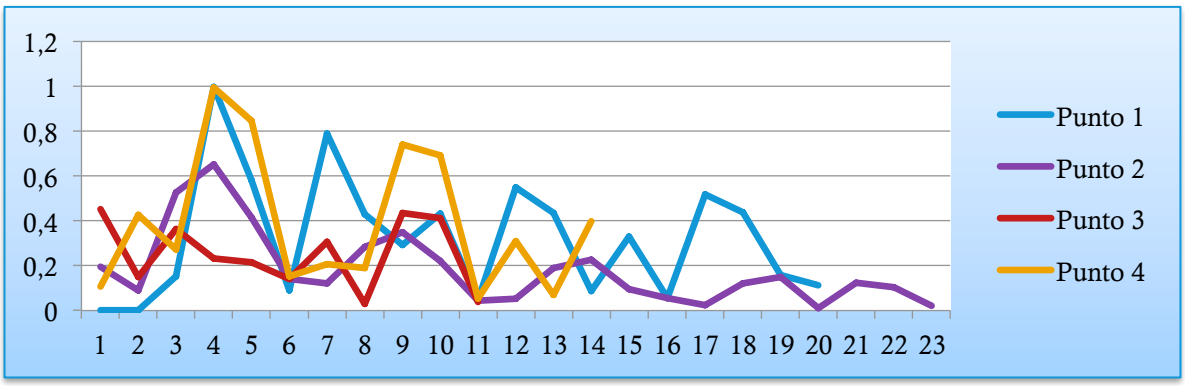
También de encuentran obviedades reflejadas en el gráfico que demuestran que la prueba realizada es correcta. Por ejemplo, aquellos puntos que están más alejados el uno de otro necesitan de más pasos para ser alcanzados (diferencia entre los pasos de 1 a 2 y de 2 a 3).

Como punto final para esta prueba se puede estipular que la prueba es válida a partir del primer punto debido a lo mencionado anteriormente sobre la adaptación del controlador. Además el resultado final es satisfactoria aunque se seguirá buscando un resultado más óptimo.

Prueba 3

Esta prueba ha sido ejecutada con las variables: *Kp = 0.9, Ki = 1.5, Kd = 0.2* y una velocidad *V = 0.*



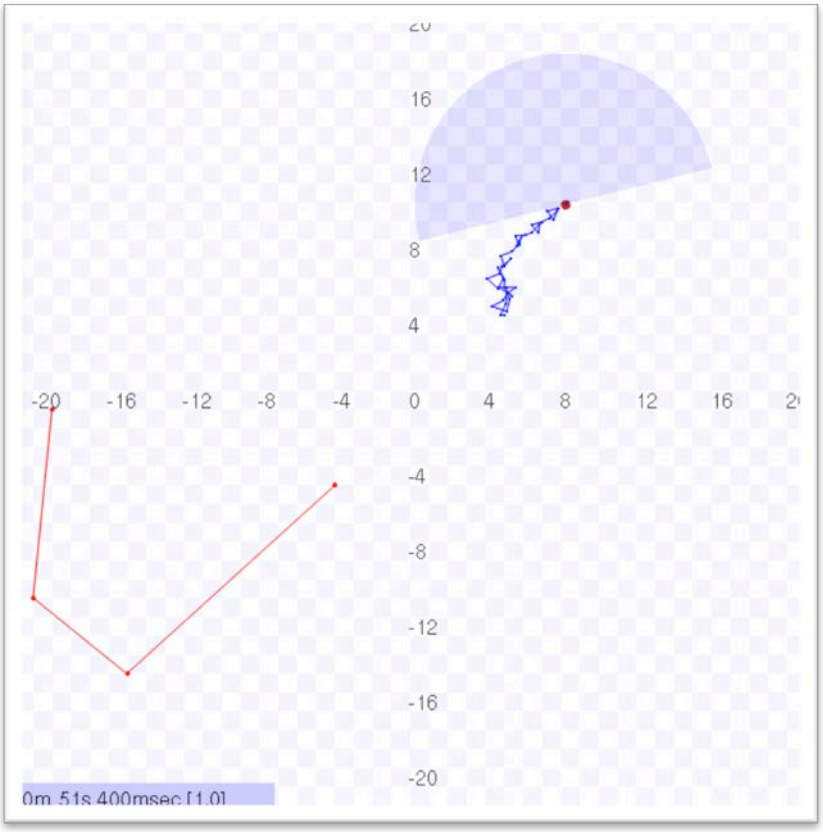


Tras el periodo de adaptación mencionado anteriormente se puede observar una gran desmejora respecto al anterior sistema ya que en los cambios de punto (más notable entre el punto 3 y 4) el sistema tiende a desestabilizarse. Esto puede ser provocado por la variable *Kd*. Dado que la variable derivativa es aquella que acumula el error cometido anteriormente para prevenir próximos errores se puede suponer que dicha variable debe ser nula o casi nula ya que a la hora de cambiar de punto estos valores no son útiles.

Además de la desestabilización del sistema al a entrada el punto, el sistema tarda muchos pasos en volver a la trayectoria ideal.

Prueba 6

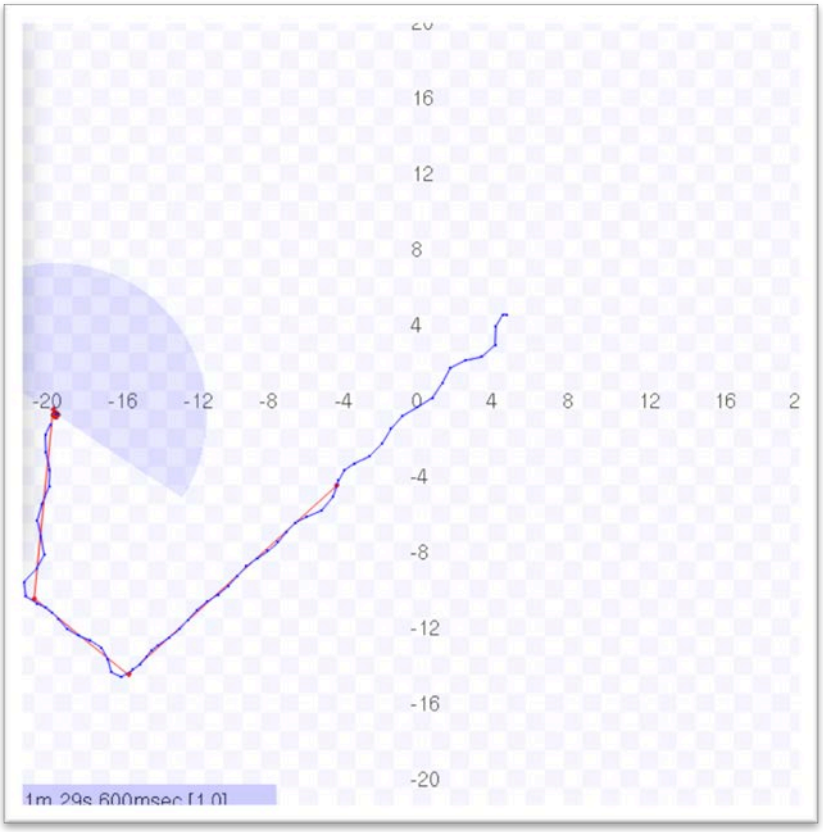
Esta prueba ha sido ejecutada con las variables: *Kp = 0.9, Ki = 1.5, Kd = 0.5* y una velocidad *V = 0.*

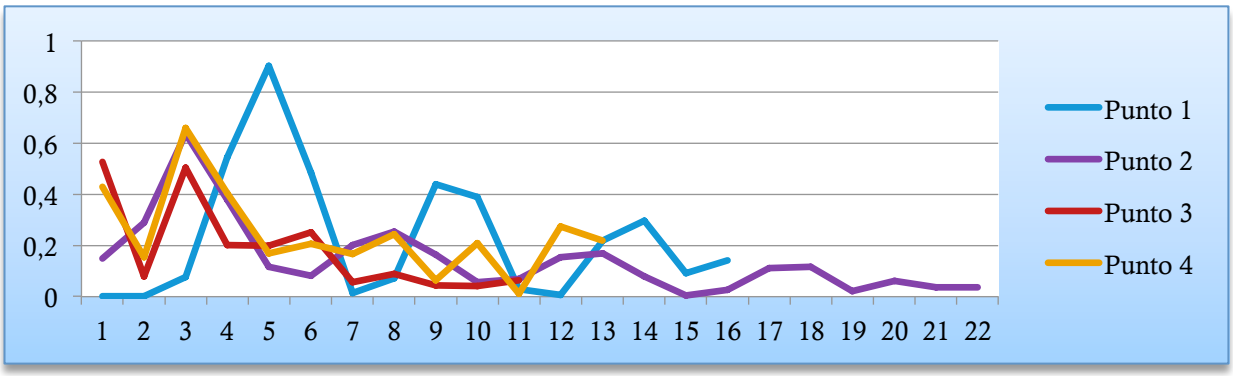


Para ahondar más en la idea de que la variable *Kd* no aporta valor al controlador en este sistema, la prueba 6, con un valor *Kd* = 0.5, se colapsa perdiendo completamente la trayectoria ideal y dando vueltas en círculos. Esta prueba no tiene gráfica ya que la imagen refleja fielmente lo comentado anteriormente.

Prueba 10

Esta prueba ha sido ejecutada con las variables: *Kp = 0.9, Ki = 1, Kd = 0* y una velocidad *V = 0.7*.





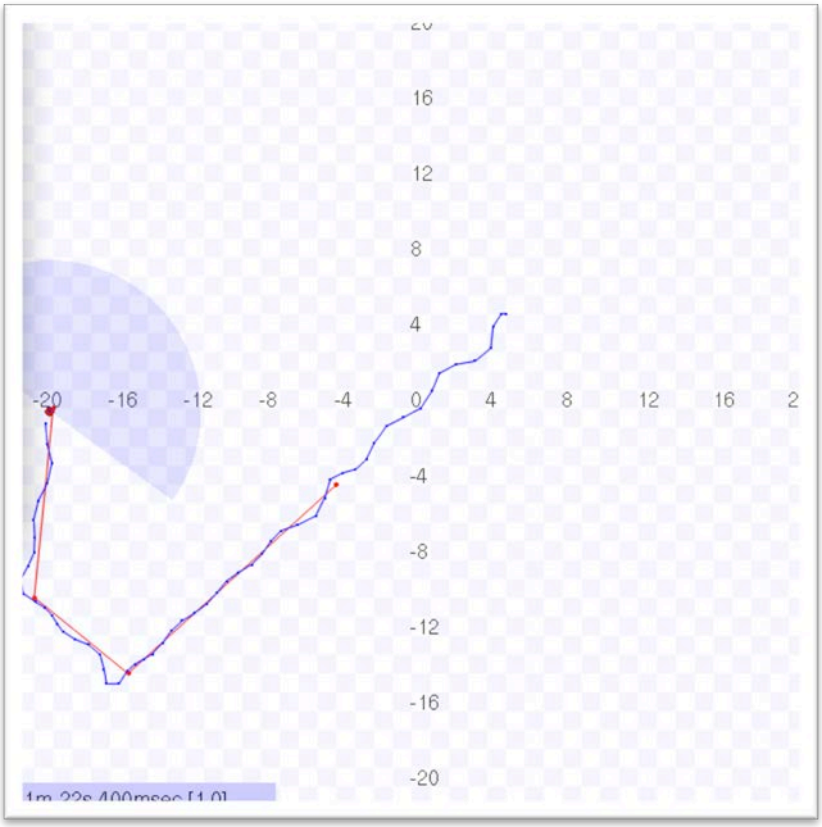
Esta prueba es la que más se acerca al sistema óptimo de las realizadas durante este recorrido. Como se puede comprobar, tras el periodo de adaptación, el sistema se estabiliza realmente bien y solo sufre una leve distorsión al alcanzar un punto y variar su rumbo.

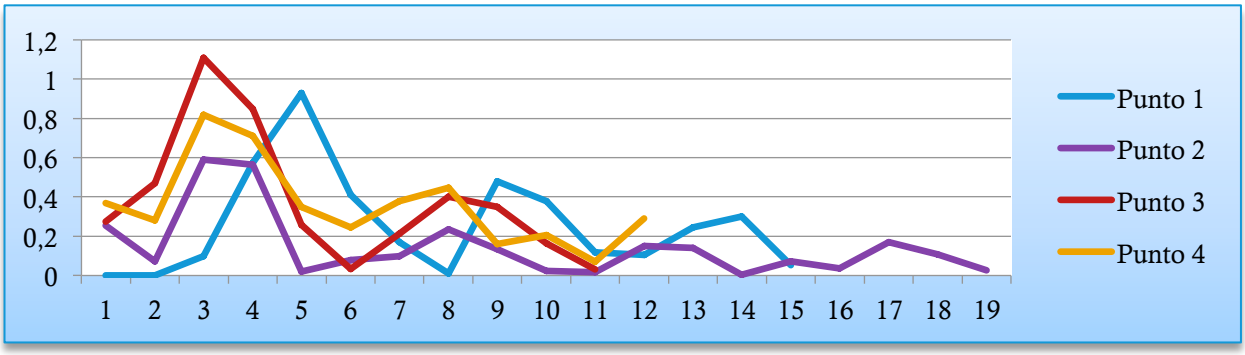
Al declarar la variable *Kd* = 0 estaremos ejerciendo un truncamiento total del componente derivativo del sistema. Esto convierte nuestro controlador en un controlador PI que actúa notablemente mejor que los que tienen esta componente.

Se puede comprobar en los resultados de la prueba 9 que se asemejan mucho a los de esta 10 y la única diferencia es la componente *Ki* que toma un valor 1.4 mientras que en la prueba 10 tiene un valor 1.

Prueba 11

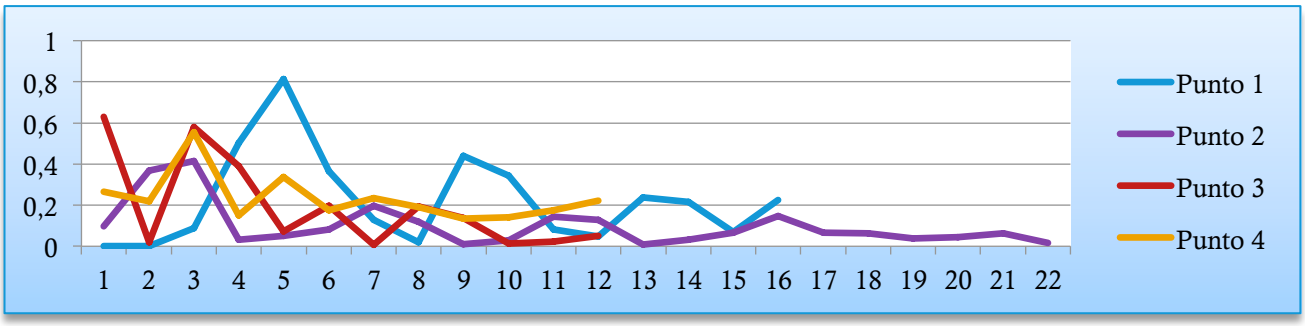
Esta prueba ha sido ejecutada con las variables: *Kp = 0.9, Ki = 1.4, Kd = 0* y una velocidad *V = 0.8.*





Esta prueba es muy similar a la realizada en la prueba 9 solo que se aumentó la velocidad 0.1 más. Se puede observar que el error durante la mayor parte del recorrido es mayor debido a que el sistema implementado trabaja más lentamente que el simulador. Por tanto, en el periodo de tiempo entre pasos, el robot ha recorrido más distancia.

Para facilitar la comprensión de este fenómeno al lector se incluye la gráfica de la prueba 9.



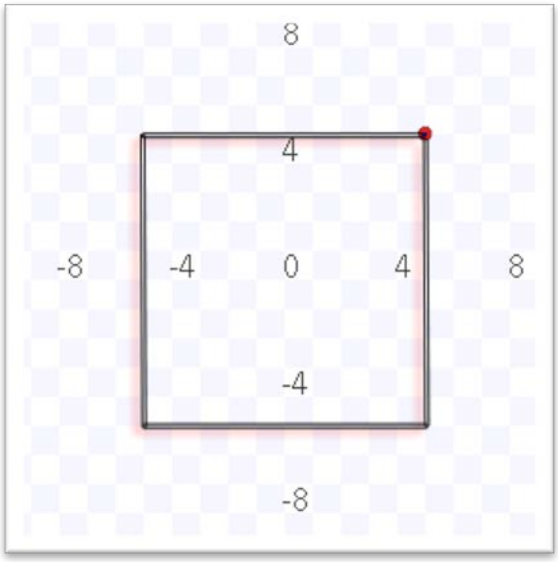
Una vez realizadas las pruebas iniciales para obtener unos valores válidos para el sistema se ejecutarán unas pruebas que determinaran si el sistema es asequible para el control de un robot simulado y, además, se tratará de obtener el resultado óptimo de las variables mencionadas anteriormente.

Pruebas en simulación.

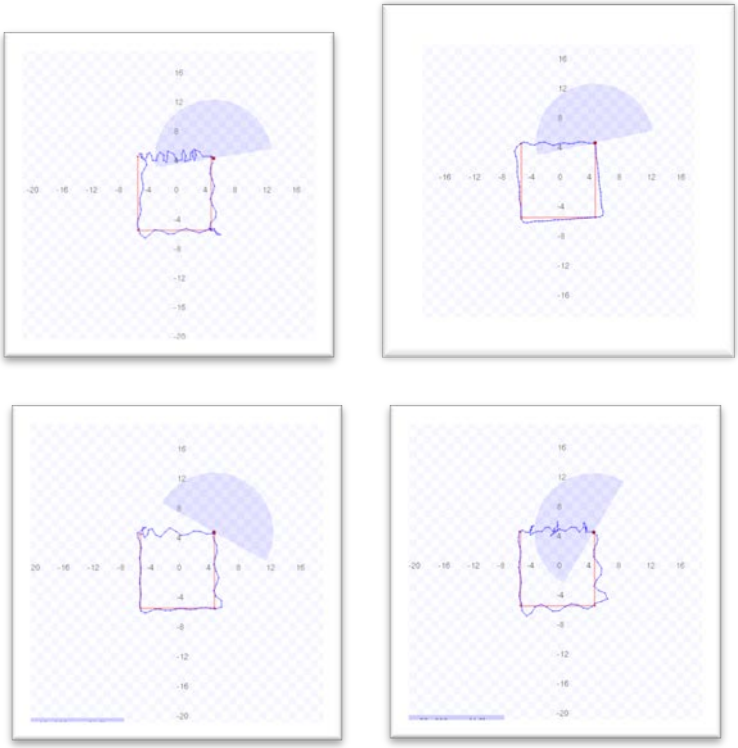
Pruebas realizadas con la trayectoria: (-5,5),(-5,-5),(5,-5)(5,5)

Prueba realizada	*V*	*Kp*	*Ki*	*Kd*
0	0,7	0,8	2	0
1	0,7	0,9	1	0
2	0,7	0,9	1,4	0
3	0,7	0,9	1	0,2

El objetivo de esta prueba es ver como se comporta el sistema ante una trayectoria de puntos lejanos con giros de 90º. La trayectoria elegida es un cuadrado de lado 10 puntos.



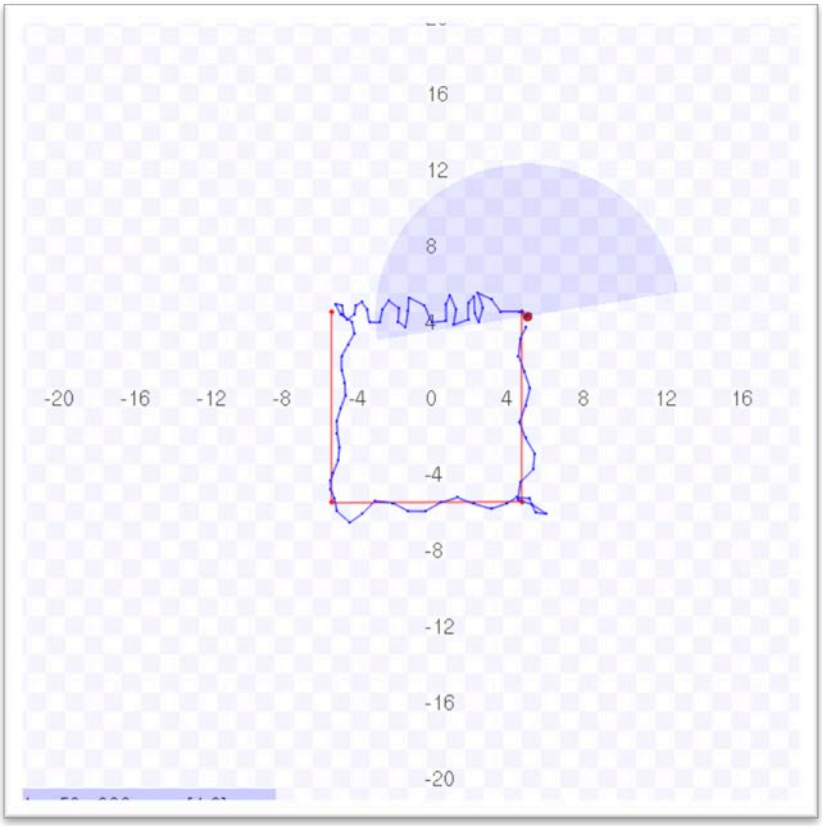
En las siguientes imágenes se puede observar las pruebas realizadas.

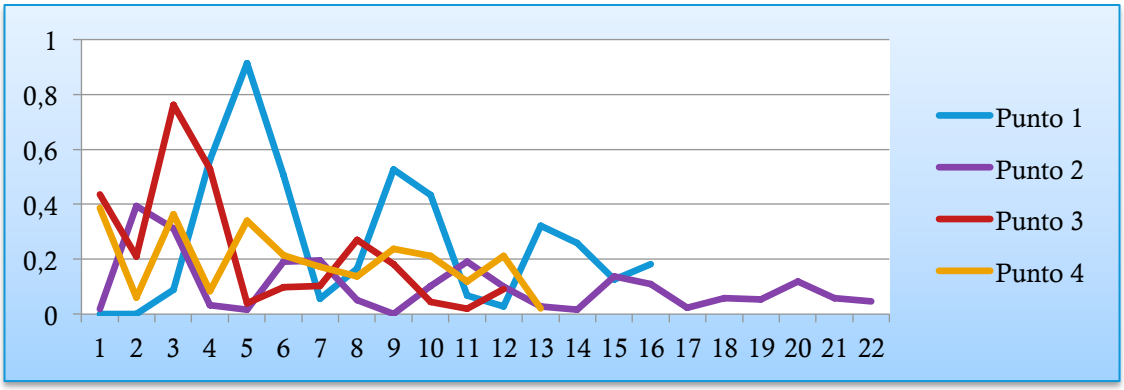


Se estudiarán en detalle aquellas pruebas que hayan resultado más interesantes para el proyecto. En este caso, las pruebas 0 y 1.

Prueba 0

Esta prueba ha sido ejecutada con las variables: *Kp = 0.8, Ki = 2, Kd = 0* y una velocidad *V = 0.7*.

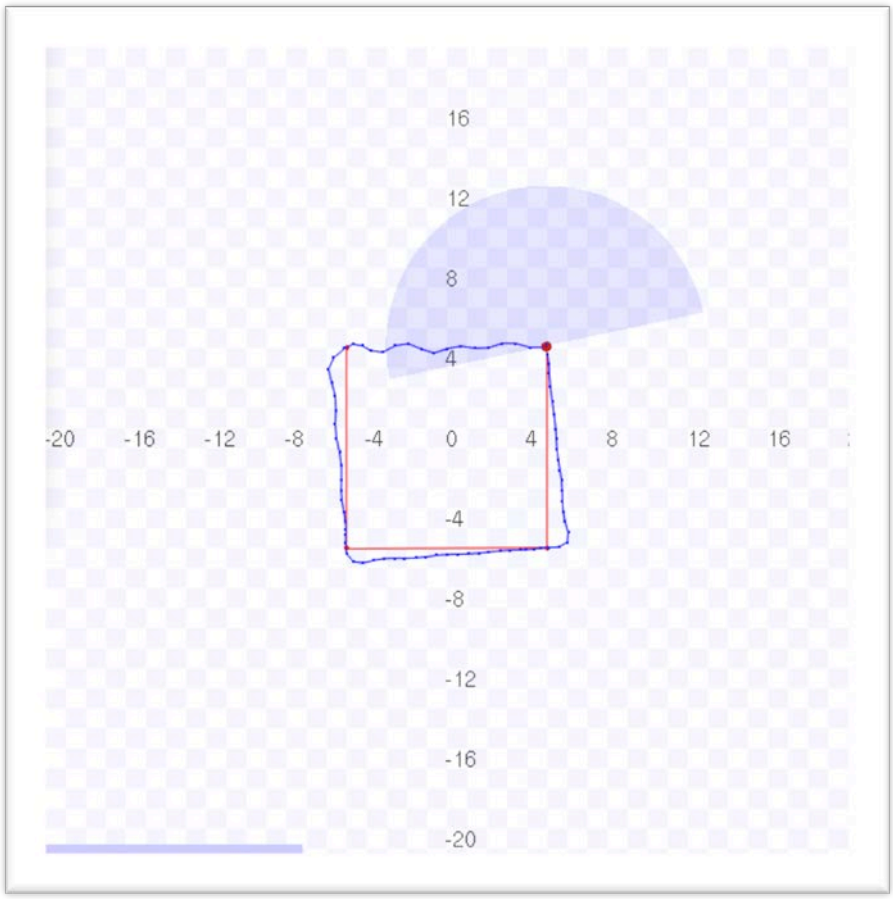


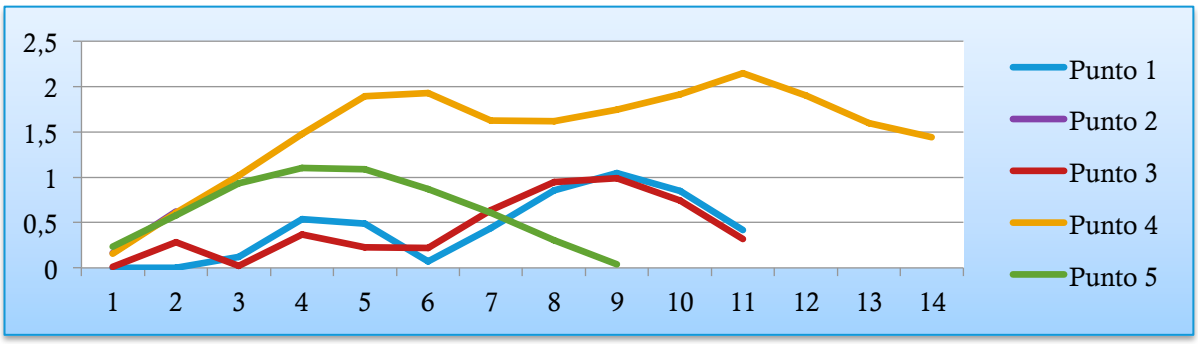


Como se puede comprobar, al igual que la anterior prueba, el sistema es bastante inestable con las variables elegidas. En la siguiente prueba se podrá observar la mejoría.

Prueba 1

Esta prueba ha sido ejecutada con las variables: *Kp = 0.9, Ki = 1, Kd = 0* y una velocidad *V = 0.7.*



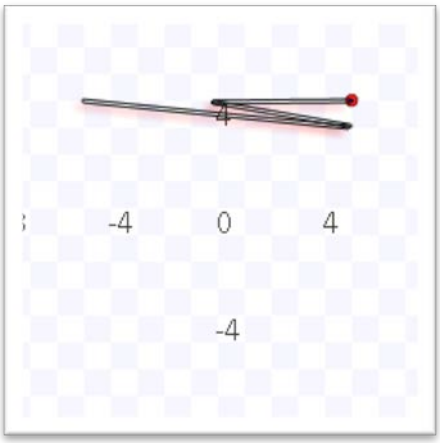


El sistema es claramente más estable que el ejemplo anterior ya que el error dismunyte con el tiempo pese a que en el punto 4 se dispara. Además, el sistema tarda menos ciclos en alcanzar los objetivos lo que le convierte en una alternativa mucho más factible que la anterior.

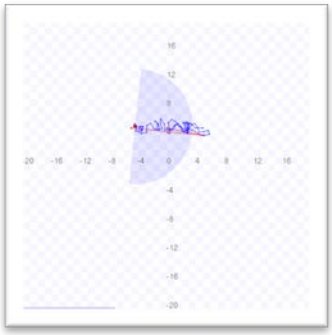
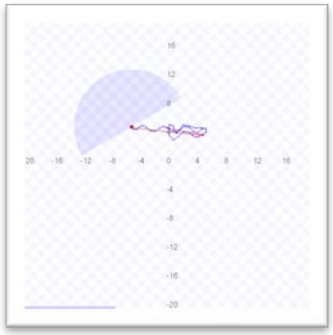
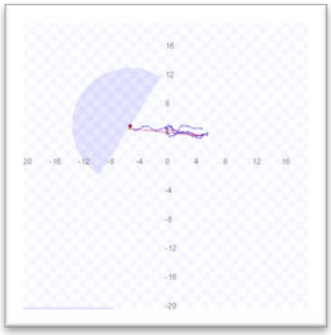
Pruebas realizadas con la trayectoria: (0,5),(5,4),(-5,5).

Prueba realizada	*V*	*Kp*	*Ki*	*Kd*
0	0,7	0,8	2	0
1	0,7	0,9	1	0
2	0,7	0,9	1,4	0

El objetivo de esta prueba es ver como se comporta el sistema ante una trayectoria de puntos con giros muy cerrados.



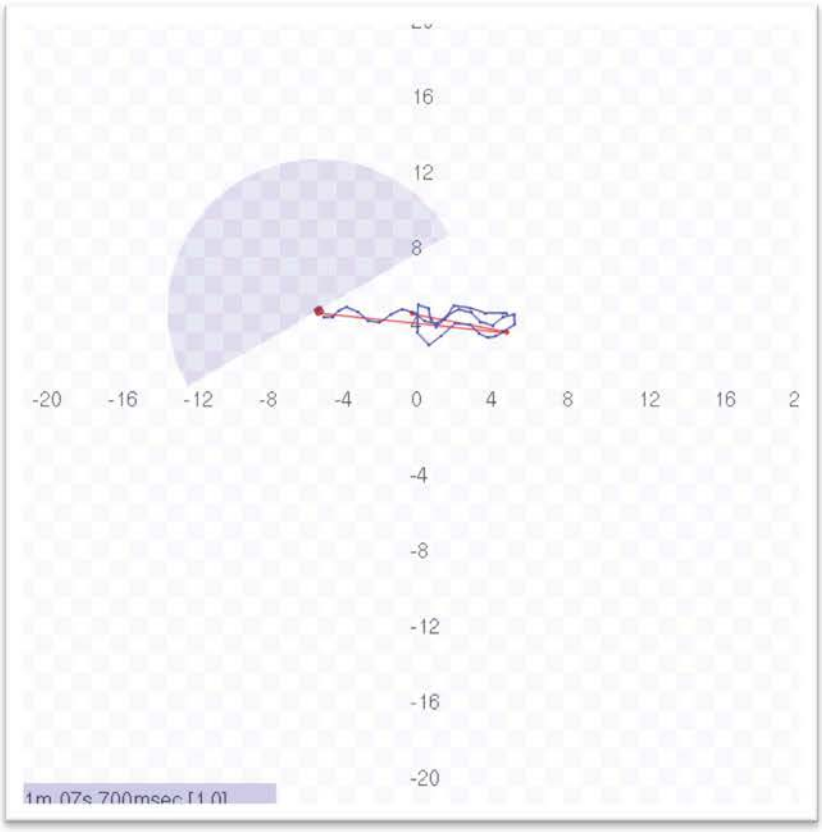
En las siguientes imágenes se puede observar las pruebas realizadas.

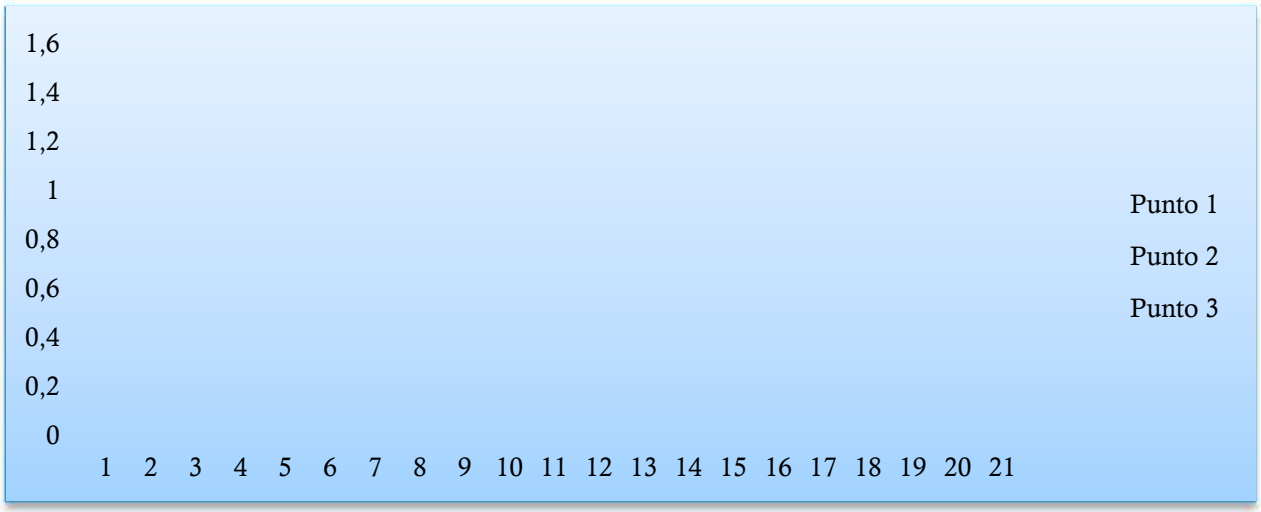




Se estudiarán en detalle aquellas pruebas que hayan resultado más interesantes para el proyecto. En este caso, la prueba 1.

Prueba 1

Esta prueba ha sido ejecutada con las variables: $Kp = 0.9$, $Ki = 1$, $Kd = 0$ y una velocidad $V = 0.7$.





Se ha elegido esta prueba como relevante ya que certifica, una vez más, que los parámetros utilizados en ella son los óptimos para este sistema. Se demuestra que da igual la trayectoria a realizar, estos parámetros adaptan el sistema para obtener un resultado óptimo.

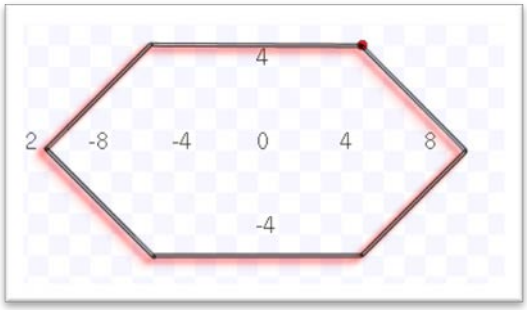
Tal y como se esperaba, si el robot tiene como dirección inicial la dirección de la trayectoria ideal el sistema tiende a un error cero en los primeros pasos. Al hacer un pequeño giro el sistema se desequilibra y tarda algunos pasos en equilibrarse.

Como es previsible, ante giros cerrados el sistema comete un error más alto y fluctúa hasta volver a la dirección de trayectoria ideal. Esto se puede observar porque al principio de cada punto el sistema tiene un error muy alto y luego baja notablemente.

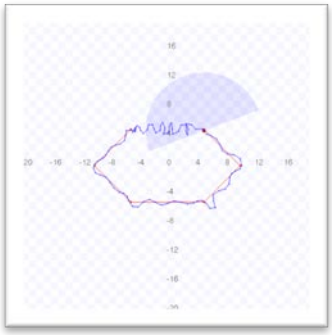
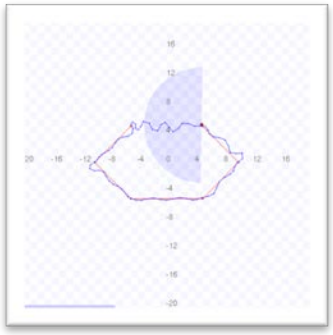
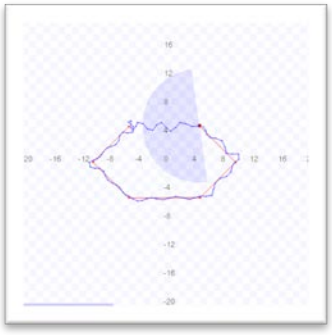
Pruebas realizadas con la trayectoria: (-5,5),(-10,0),(-5,-5),(5,-5),(10,0),(5,5).

Prueba realizada	*V*	*Kp*	*Ki*	*Kd*
0	0,7	0,8	2	0
1	0,7	0,9	1	0
2	0,7	0,9	1,4	0

El objetivo de esta prueba es ver como se comporta el sistema ante una trayectoria de puntos con giros muy cerrados y abiertos alternativamente. La trayectoria elegida es un polígono de seis lados donde podemos observar vértices muy cerrados otros muy abiertos.



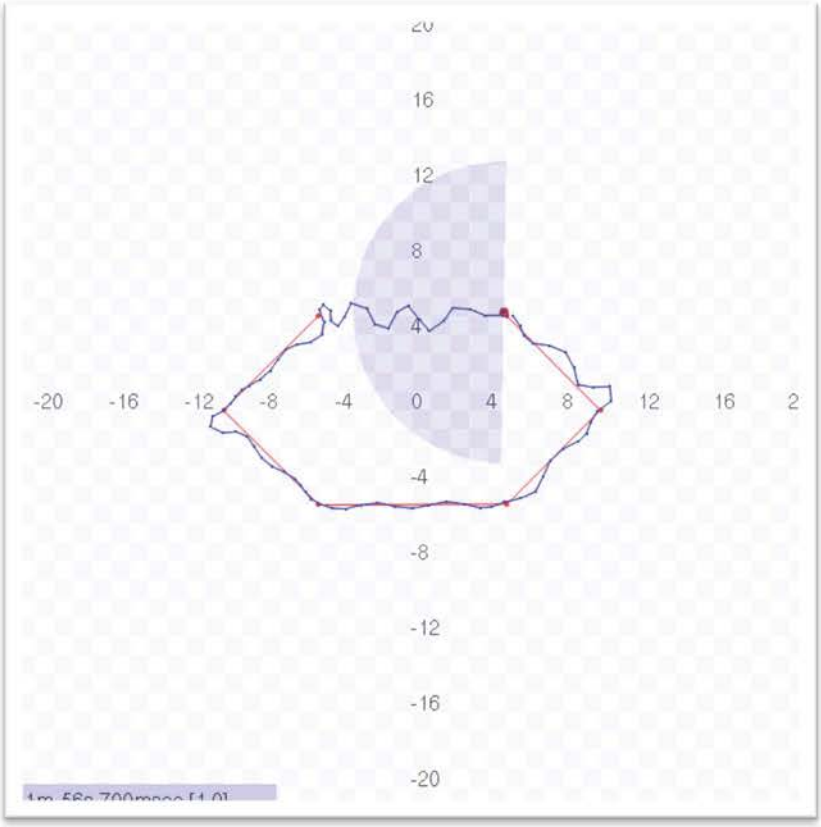
En las siguientes imágenes se puede observar las pruebas realizadas.

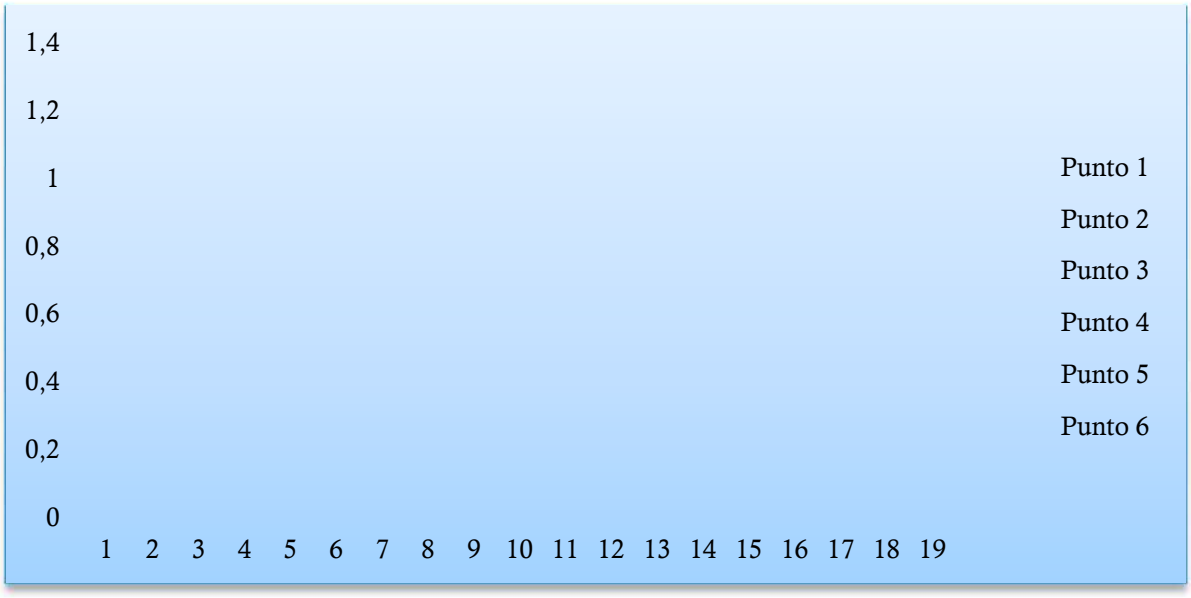




Se estudiarán en detalle aquellas pruebas que hayan resultado más interesantes para el proyecto. En este caso, la prueba 1.

Prueba 1

Esta prueba ha sido ejecutada con las variables: *Kp = 0.9, Ki = 1, Kd = 0* y una velocidad *V = 0.7.*



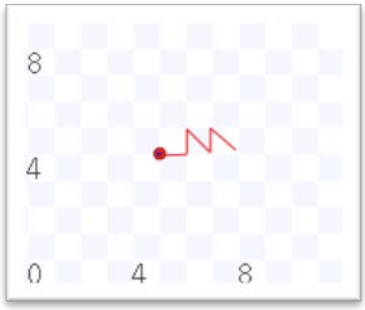


Tal y como se esperaba, en los punto en los que el sistema debe realizar giros cerrados (al principio del punto 2 y el punto 6) los picos de error son más altos. En los giros más abiertos (es decir, al principio del resto de los puntos) las fluctuaciones no son tan alas y el resultado, como se puede ver en la imagen es que los giros se realizan de manera bastante suave. Llama la atención observar como, al acercarse al punto final, el sistema fluctúa dando dos picos de error altos. Esto puede ser considerado un error del dispositivo de simulación (provocado a conciencia) que genera un giro deficiente y, por lo tanto, un pico en el error.

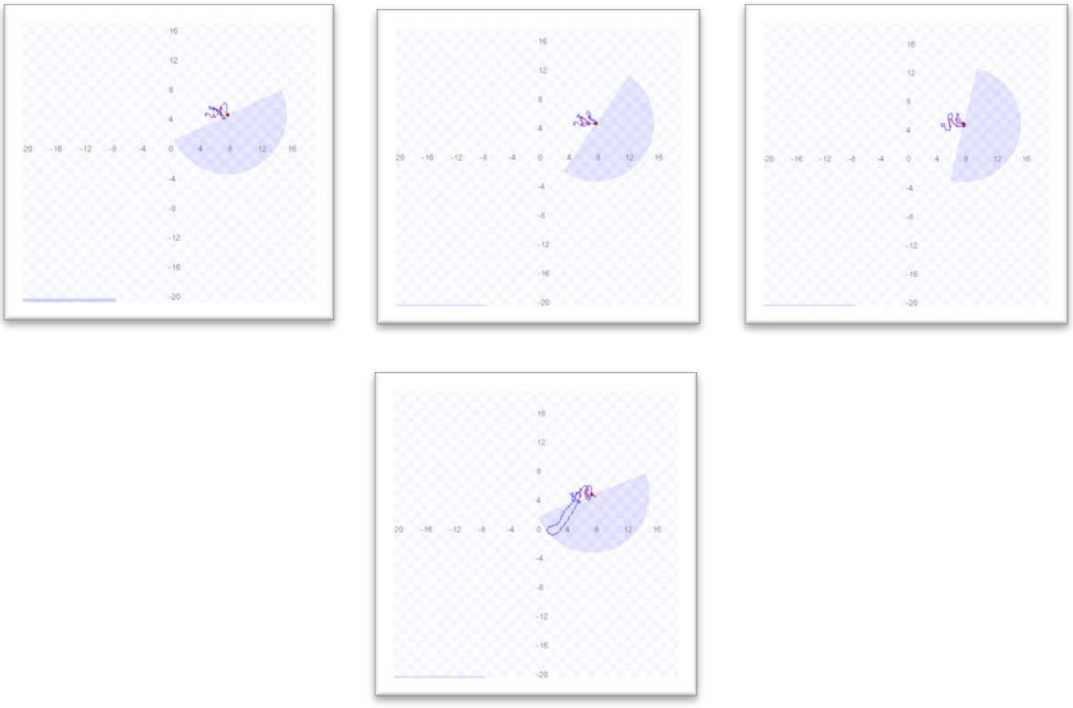
Pruebas realizadas con la trayectoria: (6,5),(6,6),(7,5),(7,6),(8,5).

Prueba realizada	*V*	*Kp*	*Ki*	*Kd*
0	0,7	0,9	1	0
1	0,7	0,9	1	0,1
2	0,7	0,9	1	0,3
3	0.7	0,9	1	0,5

El objetivo de esta prueba es comprobar como se comporta el sistema ante giros muy cortos y pronunciados. Se tratará de aplicar un coeficiente *Kd* más elevado que en las anteriores pruebas para ver si este interfiere positivamente en el comportamiento del sistema.



En las siguientes imágenes se puede observar las pruebas realizadas.

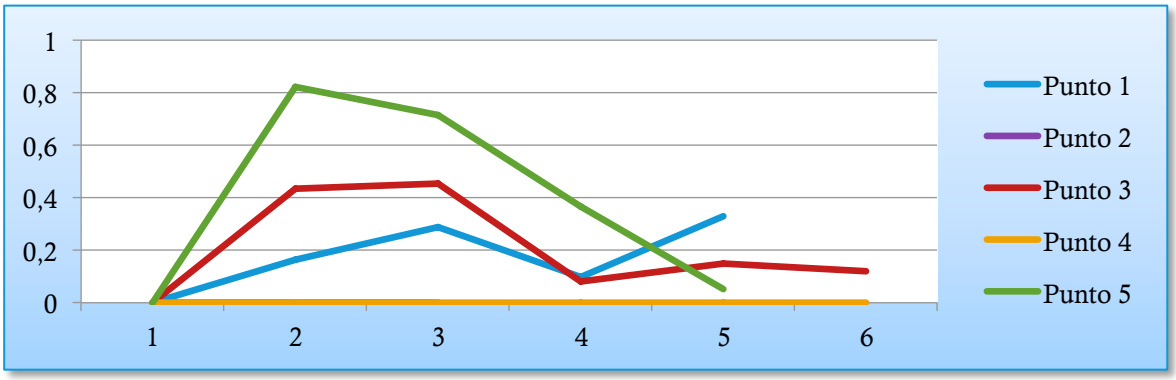


Se estudiarán en esta prueba tres de los casos de manera paralela para observar cuales son las diferencia en los errores.

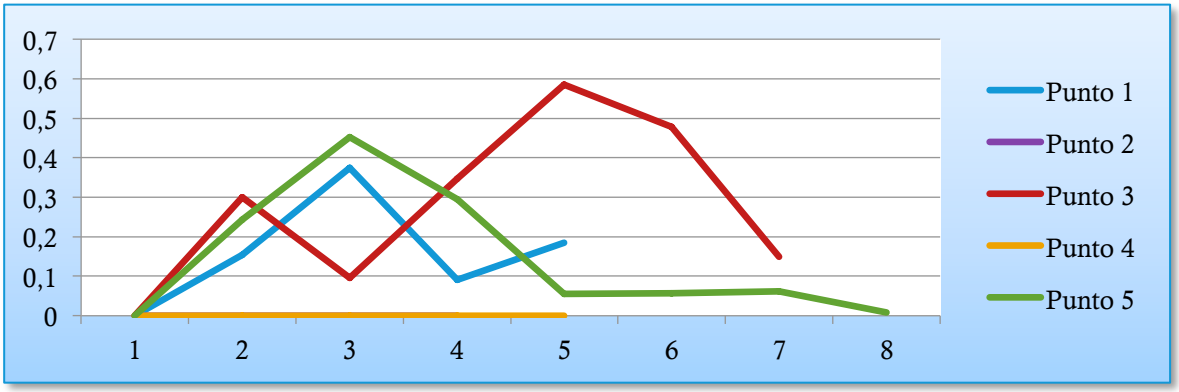
En las gráficas han se puede observar que muchos valores de los errores son cero. Esto ha sido cambiado a cero ya que el sistema, al calcular el error, devolvía *nan*

como resultado ya que la división de la distancia del punto a la trayectoria ideal era un número muy próximo a cero. Para poder sacar conclusiones de estos datos, se ha decidido colocar como 0 dichos valores.

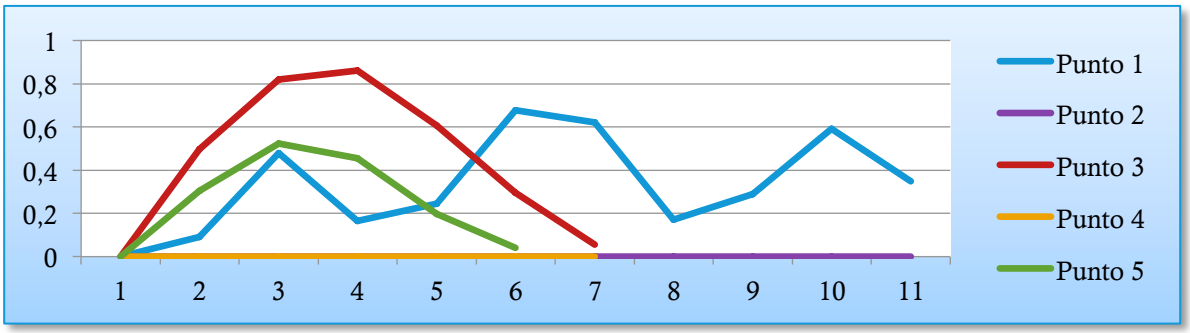
Con Kd = 0



Con Kd = 0.1



Con Kd = 0.5



Si se observa la primera gráfica comprobamos que el sistema se va desequilibrando si el coeficiente *Kd* es 0 ya que los giros deben ser muy rápidos y violentos. Tan solo con aumentar dicho coeficiente en 0.1 se aprecia que el error se va estabilizando (excepto en algunos picos donde el error sube mucho).

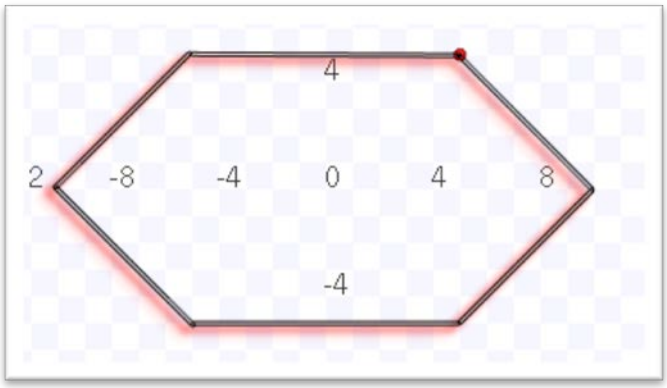
Se ha incluido el caso en el que *Kd* es 0,5 pese a que al principio es completamente inestable para ver como el sistema se equilibra mejor cuando está sobre la trayectoria alcanzando los puntos.

La conclusión final de esta prueba es que, pese a ser un sistema que va mejorando, el aumento de *Kd* no es óptimo para el sistema ni siquiera en puntos cercanos donde los giros son muy violentos.

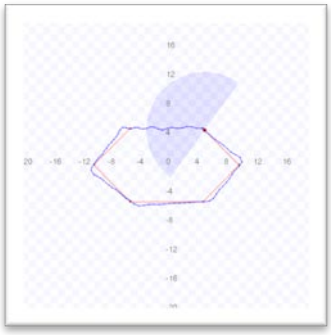
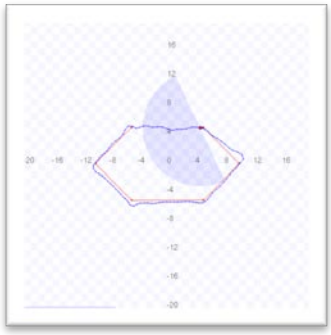
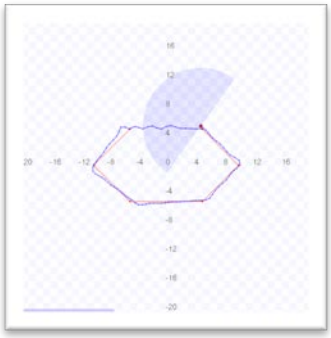
Pruebas realizadas con la trayectoria: (-5,5),(-10,0),(-5,-5),(5,-5), (10,0),(5,5) con ruido.

Prueba realizada	*V*	*Kp*	*Ki*	*Kd*	*ruido*
0	0,7	0,9	1	0	0
1	0,7	0,9	1	0	0,5 aleatorio
2	0,7	0,9	1	0	0,5 no aleatorio

El objetivo de esta prueba es comprobar como afecta el ruido a unas pruebas ya realizadas observando si aumenta o disminuyen significativamente las medidas de error del sistema.

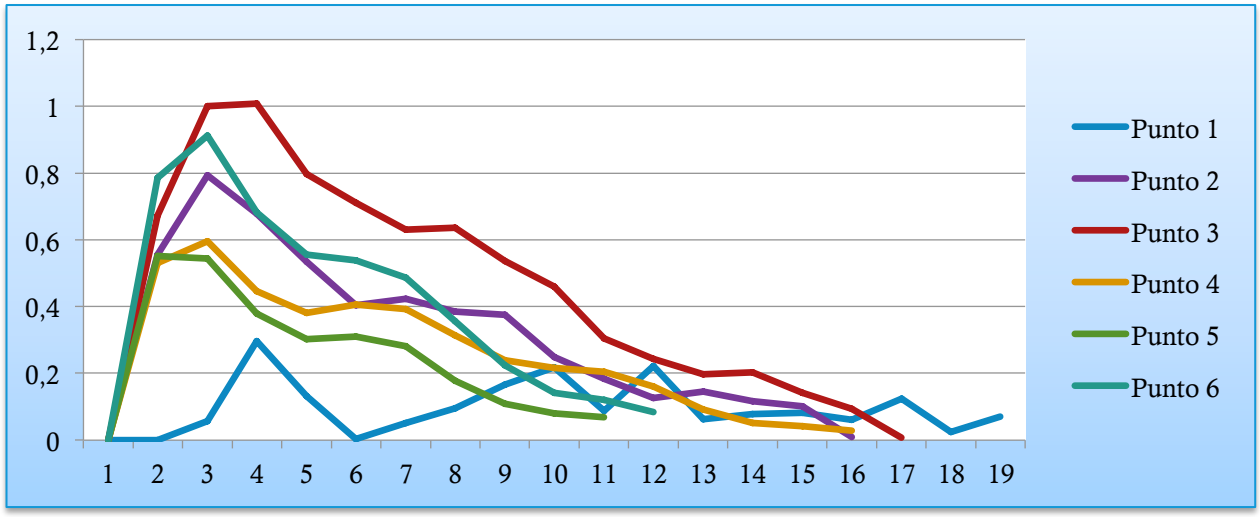


En las siguientes imágenes se puede observar las pruebas realizadas.

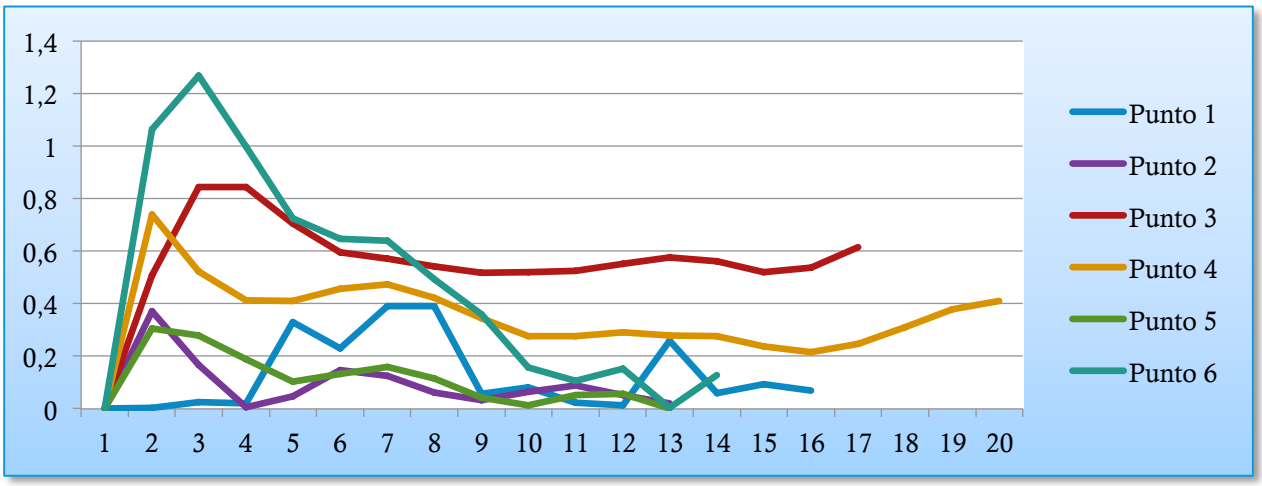




Se estudiarán en esta prueba los tres casos de manera paralela para observar cuales son las diferencia en los errores.

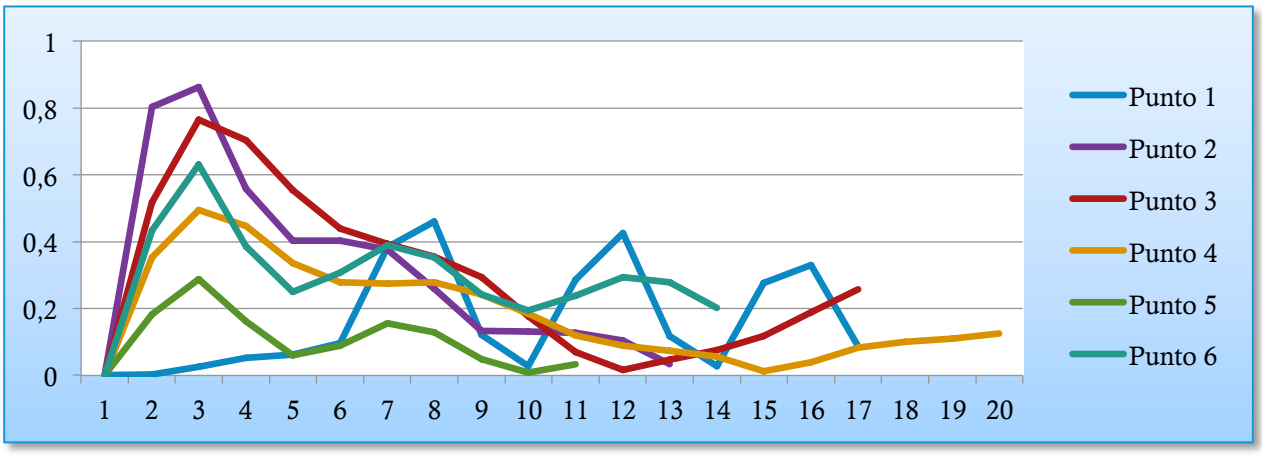
Sin ruido



Ruido Aleatorio



Ruido No Aleatorio



Tras observar las gráficas se puede comprobar que en el caso del ruido aleatorio el sistema se desestabiliza durante casi todo el recorrido sin llegar a alcanzar un valor bajo y óptimo.

Sin ruido, el sistema se mantiene tal y como se predice: disminuyendo el error hasta alcanzar la coordenada objetiva. Sin embargo, cuando el sistema posee un ruido no aleatorio (que implica que el error se genera una vez y luego se aplica siempre con el mismo valor) este baja paulatinamente aunque no llega a converger tanto como el que no posee ruido. Esto se debe a que el sistema, cuando tiene un error aleatorio, no puede predecir cual será el valor del siguiente error.

El ruido no aleatorio puede equipararse a un defecto en el dispositivo que capta la posición y, por lo tanto, genera siempre el mismo error. Como puede observarse a convergencia del error en el caso del ruido aleatorio es mucho menor que la del ruido no aleatorio y eso es porque el controlador se auto regula para estabilizarse.

Conclusiones finales del sistema simulado.

Tras haberse realizado varias pruebas con el sistema simulado, utilizando diferentes valores de los coeficientes de controlador *PID*, se puede establecer, sobre el sistema simulado de un robot con navegación automática con la señal de entrada obtenida de la posición, el coeficiente *Kd* cercano a nulo mejora los resultados de las pruebas ejecutadas.

La misión de este coeficiente es ayudar a ejecutar movimientos rápidos que necesiten un cambio de orientación pronunciado y, tal como se muestra en las pruebas, ese efecto se consigue. Pero debido a la imposibilidad de estabilizar el sistema desde el principio, no es un caso válido y es mejor descartar dicho coeficiente.

Además, con trayectorias afectadas por el ruido en los datos de entrada, el sistema se comporta peor que sin dicho ruido y, en el caso de ser aleatorio, el sistema tarda en estabilizarse más que si es un error constante en los datos de entrada.

Pruebas con del dispositivo real de video

Introducción

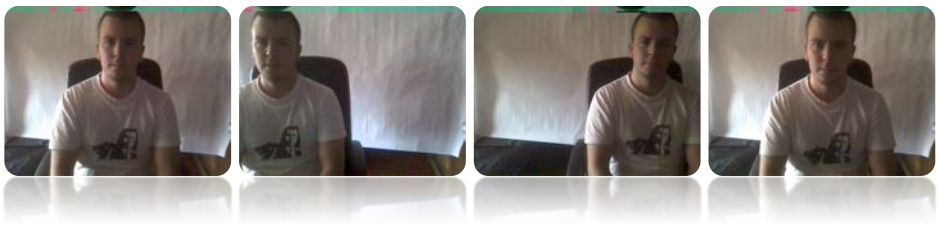
Las pruebas realizadas con este dispositivo no tienen como objetivo buscar los valores del controlador que mejor vayan a la hora de realizar un control automático del sistema. El objetivo de estas pruebas no son otras que demostrar que el sistema reacciona bien a los cambios de dirección y que el controlador interpreta correctamente los datos recibidos.

Los videos analizados son tres:

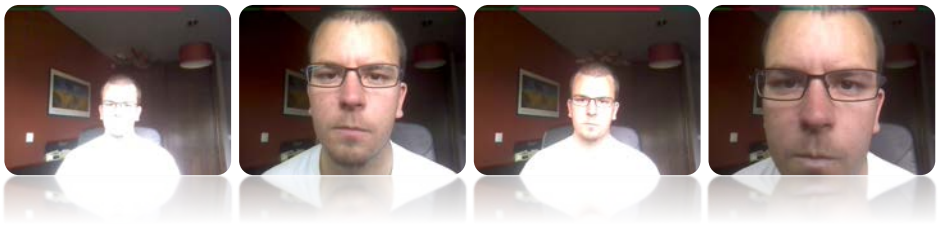
- Un video para analizar la dirección.
- Un video para analizar si se detiene el sistema al no encontrar una cara o al estar demasiado cerca.
- Un video que combina ambos objetivos.

A continuación se mostrarán detalles de los videos analizados:

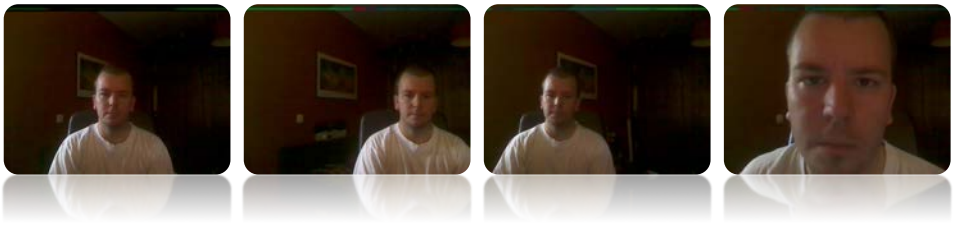
Video 1:

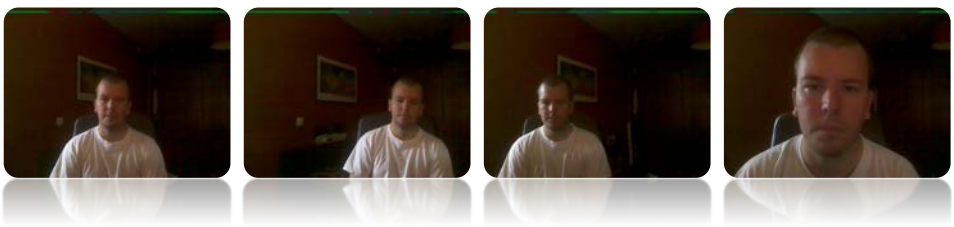


Video 2:



Video 3:





Como se puede comprobar, en el primer video la cara a detectar se mueve del centro a la derecha (desde el punto de vista de la cara), luego a la izquierda y luego vuelve al centro.

En el segundo video el individuo se mueve hacia delante, luego hacia atrás y vuelve hacia adelante.

El tercer video es una combinación de ambas tareas.

A continuación se mostrarán las gráficas de las pruebas realizadas acompañadas de los fotogramas clave de cada video.

Pruebas con el primer video: movimientos de izquierda a derecha.

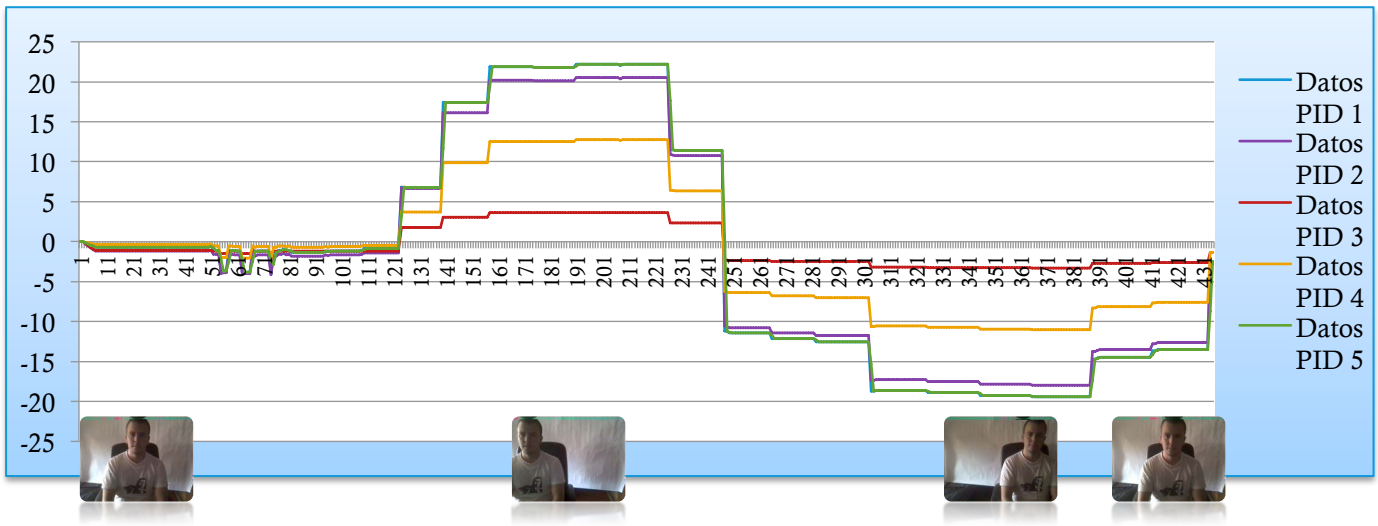
El video contiene aproximadamente 500 frames que son analizados uno por uno por el sistema implementado.

Si el sistema funciona correctamente deberíamos obtener una gráfica que de devolviese giros positivos cuando la cara detectada está a la izquierda y giros negativos cuando esta está a la derecha.

Para esta prueba se han seleccionado los siguientes valores en el controlador PID.

	Kp	*Ki*	*Kd*
Datos PID 1	0,9	2	0
Datos PID 2	0,8	2	0
Datos PID 3	0,1	1	0
Datos PID 4	0,5	0,5	0,5
Datos PID 5	0,9	1	0,5

Cada línea de cada color del gráfico equivale a una prueba realizada con cada uno de los datos del PID expuestos en la tabla. En esta ocasión se superponen las pruebas sobre el mismo gráfico para comprobar si el sistema resuelve el movimiento según lo predicho.



Tal y como se había predicho, el sistema actúa correctamente ante los giros proporcionados.

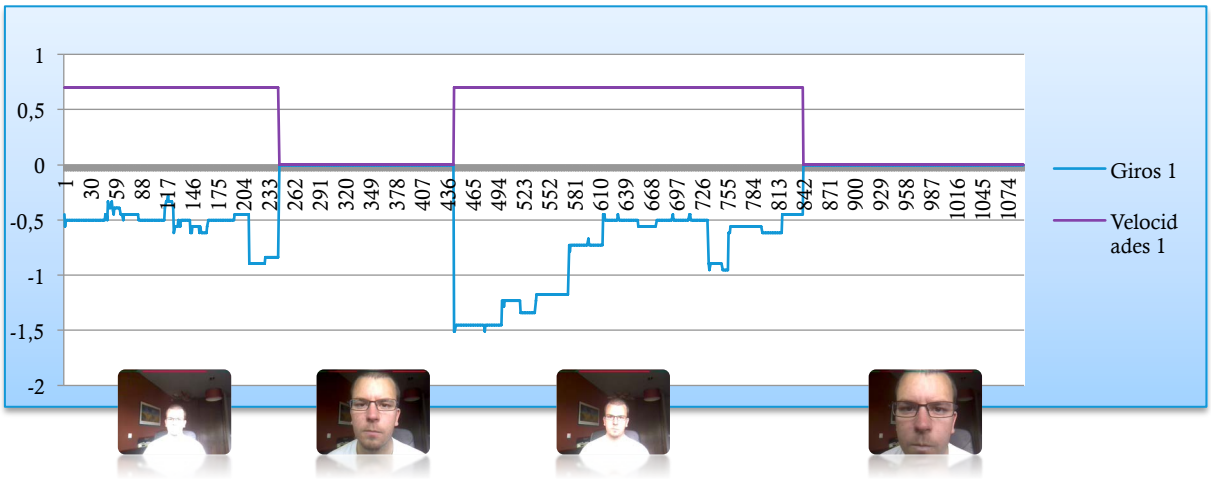
Se puede observar también que, a mayor valor de *Kp*, mayor es el giro que prevé el controlador y por tanto, es más violento.

Como conclusión de esta prueba, el sistema funciona correctamente ante los diferentes giros que proporciona el sistema de video.

Pruebas con el segundo video: movimientos acercándose al individuo.

Para esta prueba se han acumulado los valores de velocidad y movimiento a realizar para comprobar que el sistema se detiene cuando se encuentra demasiado cerca de la cara detectada o no encuentra ninguna.

El video analizado tiene alrededor de 1000 frames y retrata dos acercamientos a cámara seguidos.



Como muestra el video, podemos observar que al acercarse la cara el sistema se detiene generando una velocidad de valor *0* y un giro de valor también *0*. Esto demuestra que el sistema si funciona correctamente.

Pruebas con el tercer video: combinación de movimientos.

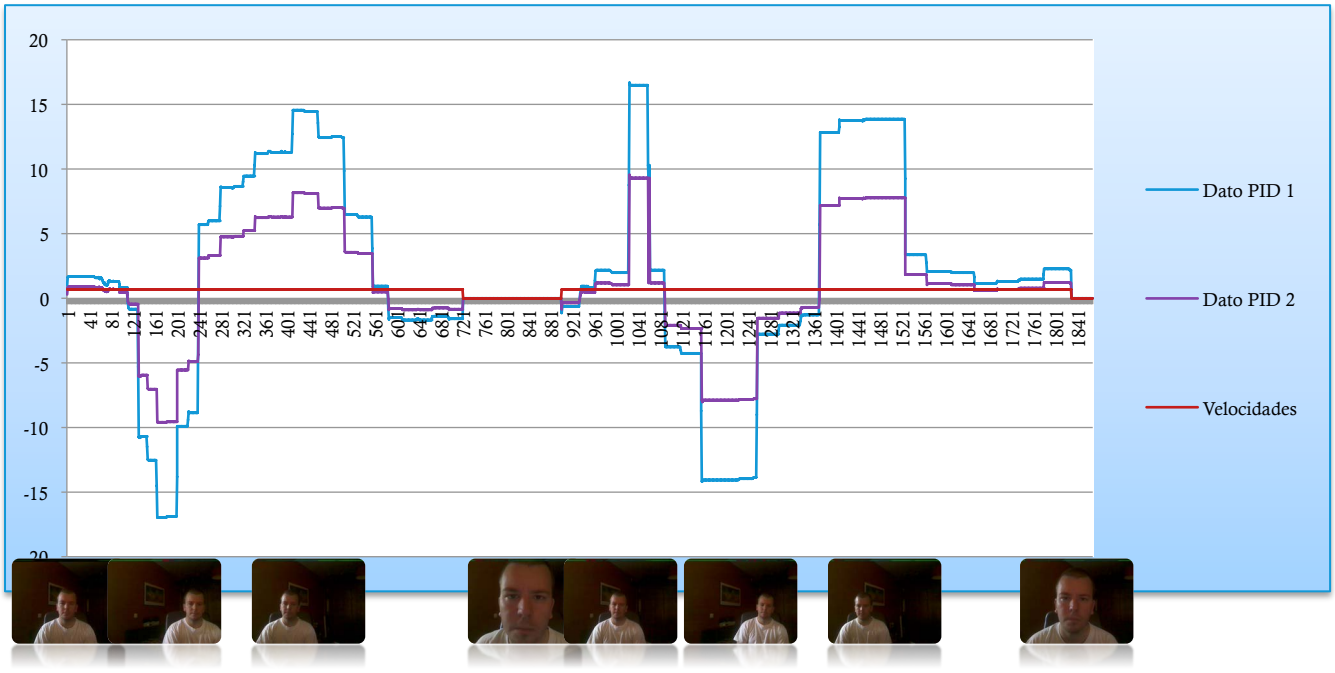
El video contiene aproximadamente 1800 frames que son analizados uno por uno por el sistema implementado.

Si el sistema funciona correctamente deberíamos obtener una gráfica que de devolviese giros positivos cuando la cara detectada está a la izquierda y giros negativos cuando esta está a la derecha. Además, cuando la cara detectada se acerca a la cámara el sistema debe detectarlo y deberá pararse.

Para esta prueba se han seleccionado los siguientes valores en el controlador PID.

	Kp	*Ki*	*Kd*
Datos PID 1	0,9	1	0,5
Datos PID 2	0,5	0,5	0,5

Cada línea de cada color del gráfico equivale a una prueba realizada con cada uno de los datos del PID expuestos en la tabla exceptuando la que recibe el valor de la velocidad que es común en ambos casos y representa si se mantiene el movimiento cuando la cara detectada está demasiado cerca o, si por el contrario, el sistema reacciona correctamente y se detiene. En esta ocasión se superponen las pruebas sobre el mismo gráfico para comprobar si el sistema resuelve el movimiento según lo predicho.



Pese a que los valores del parámetro velocidad no se mueven en la misma escala se ha decidido incluirla en este gráfico para tener una visión global y temporal de cómo el sistema se ha comportado. Tal y como se aprecia, el sistema reacciona correctamente a los giros los acercamientos.

Conclusiones

El proyecto llega a su fin tras haber realizado las pruebas con los sistemas en simulación y el dispositivo real. Tras el análisis de las pruebas de manera individual se puede declarar que los objetivos para los que este proyecto había sido ideado han sido superados.

Al principio de este documento se detallaba la importancia de la navegación en vehículos autómatas. Se pudo percibir que los dispositivos de los que hacen uso los robots para la navegación son el punto clave para su localización pero que sin un software de control útil y eficaz es imposible manejar dichos dispositivos. La intención de este proyecto era acercar al lector o al desarrollador a una solución abierta a la problemática de control de datos de sensores y generar una entrada al robot coherente con los datos que proporcionan los dispositivos.

Desde la fase de análisis del sistema objetivo, estudio de proyectos anteriores, conocimientos teóricos a adquirir, diseño del sistema final y su implementación han sido llevadas a cabo con el objetivo primordial de obtener un sistema útil y escalable que presentara una línea de trabajo clara y posibles bifurcaciones de la misma hacia objetivos más grandes y suculentos.

La mayor desventaja a la que se ha enfrentado este proyecto ha sido a la imposibilidad de probarlo con un robot real debido a la cantidad de problemas que surgieron con dicho dispositivo. Si algo merma la alegría del autor de este proyecto es la imposibilidad de ser él el que traslade este sistema simulado o semi-simulado (debido a la implementación de un dispositivo real) al robot para hacer funcionales sus objetivos.

Otra posible desventaja es la molesta y constante carga de actualizaciones que plataformas como *Player/Stage* someten a sus entornos de trabajo. El mayor problema de estas actualizaciones es que no suelen ser retro-compatibles y, por tanto, muchas de las implementaciones realizadas no funcionan con versiones de *Player/ Stage* superiores.

Tomando un carácter más positivo, el sistema implementado ha sido desarrollado sobre fuertes y buenos cimientos basados en proyectos muy bien trabajados como (Rebollo, 2010). Esto ha ayudado a que el proyecto que tiene en sus manos fuese viable para un equipo tan reducido de trabajo.

Respecto a los resultados obtenidos de las pruebas, se ha demostrado que el controlador *PID* es una digna solución para la problemática de control del sistema y que defiende bien los intereses del robot para alcanzar los objetivos. Además, mediante la variación de las constantes de dicho controlador se obtiene resultados muy importantes tales como que la constante *Kd* es en muchos casos inútil para el sistema provocando que este se desvíe de sus trayectorias.

Las pruebas nos muestran que los sistemas que poseen el coeficiente *Kd* desestabilizan el sistema y, por lo tanto, un valor nulo es el óptimo para este sistema. Esto transforma el controlador *PID* en un controlador *PI.*

Además, se ha demostrado que la implementación de un dispositivo real es claramente compatible con el diseño realizado durante la su correspondiente fase del proyecto.

Futuras líneas de trabajo

Este proyecto abre un sin fin de posibilidades de investigación y nuevos proyectos. El más directo que se puede encontrar es el de desarrollar el sistema con el dispositivo óptico sobre un robot real y probar su eficacia. Para ello se deberán calibrar los parámetros del controlador para crear un sistema que fuese efectivo.

Otra posible línea de desarrollo inmediata es la creación de un sistema de trayectorias con el dispositivo óptico ya implementado. Esto implicaría que el sistema no perseguiría un objetivo si no que alcanzaría una serie de puntos tal y como lo hace el sistema simulado.

Además, el sensor óptico está desarrollado con al detección de caras humanas en al imagen lo cual es bastante inútil a la hora de seguir los pasos de un individuo. Una posible ampliación de este proyecto sería que el sistema detectase colores y de esta manera pudiese seguir a alguien esté caminando.

Por otra parte, se puede extender otra rama de desarrollo hacia la integración y fusión de datos para los diferentes dispositivos que un robot real puede poseer. De esa manera el rumbo a generar será mucho más fiable y consistente.

Además de las posibles líneas de trabajo y debido a la escalabilidad de este proyecto, la inclusión de nuevos controladores, dispositivos y sensores reales o incluso nuevos objetivos para el sistema es un factor muy importante a la hora de continuar implementaciones basadas en este proyecto.

Bibliografía

Wescott, T. (October de 2000). PID without PhD. *EE Times-India* .

Ana González Marcos, F. J. (2006). *Técnicas y algoritmos básicos de visión artificial.* UNIVERSIDAD DE LA RIOJA SERVICIO DE PUBLICACIONES .

Andrei, N. (2005). Modern Control Theory, A historical perspective. *Center for Advanced Modeling and Optimization* .

Carlos Cruz Corona, J. M. (2007). Estrategias Cooperativas paralelas en la solución de problemas de optimización. *Revista Iberoamericana de Inteligencia Artificial* .

Gary Bradski, A. K. (2008). *Learning OpenCV.* O'Reilly.

Kart J. Aström, T. H. (2009). *Control PID avanzado.* PRENTICE-HALL.

Marc Raibert, K. B. (2008). BigDog,the Rough-Terrain Quaduped Robot. *Boston Dynamics* .

Ollero, A. (2001). *Robótica. Manipuladores y robots móviles.* Barcelona: Marcombo S.A.

Rebollo, T. (2010). *Diseño e Implementación de una capa de software para el control de robots mediante Player/Stage.* Madrid: Universidad Carlos III de Madrid.

Reed Hedges, K. S. (2008). *Player Project*. From http://playerstage.sourceforge.net/index.php?src=player.

Reed Hedges, K. S. (2010). *Stage Project*. From http://playerstage.sourceforge.net/index.php?src=stage

SPAIN, I. I. (2010). Vehículos aéreos no tripulados (UAVs).

Anexo I – Manuales.

Manual de instalación y uso.

Este manual explicará la configuración e instalación del entorno Player-Stage así como el software creado para este proyecto y las librerías Open CV. El objetivo de esta sección es facilitar el trabajo al lector a la hora de instalar el entorno de trabajo.

Instalación de Player 3.0.0 en el sistema operativo Ubuntu

1. Primero se han de descargar los paquetes de Player de la siguiente página:
   - http://sourceforge.net/projects/playerstage/files/Player/

   Se recomienda descargar la versión 3.0 de Player ya que es la que ha sido utilizada en este proyecto.
2. Extraer el contenido de los dos paquetes comprimidos.
3. Para instalar Playr se deberá usar el siguiente comando desde un terminal dentro de la carpeta descomprmida:

```
$ ./configure --prefix=<INSTALL_DIR> --disable-alldrivers --enable-cmvision --enable-camerav4l --enable-urglaser --enable-amcl --enable-vfh --enable-wavefront --enable-logfile --enable-mapfile --enable-mapscale --enable-vmapfile --enable-bumpersafe --enable-lasersafe
```

4. La etiqueta <INSTALL_DIR> deberá ser la localización donde se quiere que se instale el software. Si esta etiqueta se omite (junto con –-prefix) la ruta por defecto será /usr/local.
5. Una vez instalado se deberá configurar por defecto el entorno Player 3.0.0 mediante lo siguientes comandos desde el directorio donde se encuentran los archivos descomprimidos.

```
$ cd player-<version>
$ mkdir build
$ cd build
$ cmake ../
```

6. En el caso de no querer usar la ruta por defecto se deberá utilizar este otro comando.

```
$ cmake -DCMAKE_INSTALL_PREFIX=<INSTALL_DIR> ../
```

7. Una vez configurado se deberá completar la instalación con estos comandos:

```
$ make $ sudo make install
```

8. Si al tratar de usar Player tras una actualización de Ubuntu este da un error al cargar las librerías libgeos-3.0.0.so se deberá usar el siguiente comando:

```
$ cd /usr/lib $ sudo ln -s libgeos-3.1.0.so libgeos-3.0.0.so
```

Instalación de Stage 3.2.0 en el sistema operativo Ubuntu

1. Se deberá descargar el paquete de Stage de la siguiente página: https://github.com/rtv/Stage
2. Tras la descompresión de los archivos descargados, se deberá entrar dentro del directorio donde se ha descargado y ejecutar el siguiente comando apra configurarlo:

```
$ ./configure
```

3. Tras configurarlo, se deberá compilar mediante los siguientes comandos:

```
$ mkdir build
$ cd build
$ cmake ../
```

4. Tras la compilación se deberá instalar mediante los siguientes comando:

```
$ make
$ sudo make install
```

5. Si se desea guardar todos los archivos .world de ejemplo se deberá compiar el directorio worlds a una carpeta que se desee. Por ejemplo:

```
$  mkdir /home/$USER/playerstage
$  cp -r worlds/ /home/$USER/playerstage
```

Verificaciones de Stage y Player en el sistema operativo Ubuntu

1. Para verificar las instalaciones realizadas se deberá ejecutar el siguiente comando:

```
$ which player
```

2. Una vez verificado esta instalación se podrá ejecutar el código de este proyecto y verificar también Stage.

Instalación de las librerías OpenCV en Ubuntu

1. Descargar las librerías de OpenCV desde la siguiente dirección: http://sourceforge.net/projects/opencvlibrary
2. Se deberán instalar los prerrequisitos de librerías y herramientas como cmake o pkg-config si no se encuentran instaladas ya.
3. Una vez descargadas se procederá a la instlación mediante los siguientes comandos desde la carpeta donde contenemos los instaladores:

```
$  cd ~/projects/opencv
$  mkdir releáse
$  cd release
$  cmake -D CMAKE_BUILD_TYPE=RELEASE –D
  CMAKE_INSTALL_PREFIX=/usr/local -D BUILD_PYTHON_SUPPORT=ON
```

4. Una vez instalados se podrá verificar la instalación con el propio software del proyecto.

Primeros pasos y uso del software.

1. Se deberá descomprimir el archivo contenedor del software.
2. Si se desea ejecutar la interfaz gráfica debe pinchar dos veces sobre el icono de PID_GUI.sh.
3. Si se desea ejecutar desde la línea de comandos se han de abrir dos terminales y, en la carpeta descomprimida, ejecutar cada una de estas sentencias cada una en un terminal.
    a. sh play_player.sh
    b. sh play_movimiento.sh

Interfaz gráfica: Modo de empleo.

Para utilizar esta interfaz gráfica es necesario poseer la utilidad Java instalada. Si no se posee dicha utilidad se puede obtener mediante la siguiente línea de comando:

```
$  sudo apt-get install sun-java6-jdk sun-java6-jre sun-java6-plugin
sun-java6-fonts
```

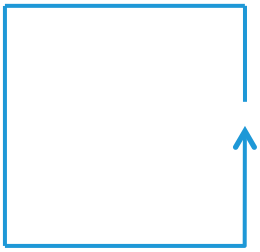
La interfaz gráfica es la herramienta que se utilizará para manejar los tres ficheros principales del programa: el fichero de trayectoria, el de variables de la simulación de este proyecto y el de variables de simulación de player/stage.

Crear una trayectoria:

- Para crear puntos en una trayectoria se deberá rellenar los campos X e Y con las coordenadas preferidas.
- Una vez añadidas, se pulsa el botón "Añadir".
- Si se desea eliminar uno de los puntos se deberá seleccionar en la lista y pulsar el botón "Eliminar".
- Una vez realizada la trayectoria se deberá pulsar el botón "Crear trayectoria". Con esta acción se genera el fichero de trayectoria que más tarde utilizará la aplicación.

Cambiar la posición inicial:

- Para cambiar la posición inicial se debe indicar la coordenada donde aparecerá inicialmente el robot y la orientación que tomará la siguiente orientación.



- Para cambiar definitivamente la posición inicial, se ha de pulsar el botón cambiar.

Cambiar las constantes del controlador PID:

- Para cambiar las variables del controlador se han de elegir mediante los valores que se pueden ver en la imagen. A parte de los expuestos anteriormente como comunes a los controladores PID se encuentran los valores *iMax* e *iMin* que son unos sesgos para que la variable i no crezca ni decrezca demasiado.
  Los valores que aparecen a la derecha del nombre de la variable indican un valor ejemplo que se podría considerar valor inicial si es la primera vez que se utiliza el sistema.

- Además de los valores del controlador se podrá añadir dos tipos de ruido al sistema:
  - Ruido aleatorio variable en cada paso que es generado cada vez que se extrae la posición del robot simulando un error no constante.
  - Ruido aleatorio que se calcula al principio de la ejecución y no se vuelve a calcular más. Este tipo de ruido puede simular un defecto en el dispositivo de simulación ya que devolverá siempre un error constante a las posiciones tomadas.

  Para indicar cual de los dos ruidos se desea añadir, solo será necesario dejar habilitada la caja de aleatorio para el primero de los ruidos o deshabilitarla para el segundo. En el caso de no querer generar ruido a la señal de posicionamiento, el valor del error deberá ser 0.
  Se recomienda no generar un ruido muy alto ya que dañaría completamente la señal provocando el colapso del sistema y, por lo tanto, nunca se alcanzaría a terminar la trayectoria deseada.
- Una vez realizados todos los cambios deseados se deberá pulsar el botón "Cambiar parámetros" para generar el fichero que más tarde leerá el programa de simulación.

Leer el fichero de errores obtenido:

- Para leer el fichero de errores obtenidos se podrá leer pulsando el botón "Fichero de errores en Excel" o "…en texto" que utilizan los programas Libre Ofice y Gedit consecutivamente.

- Si el programa se está ejecutando desde una versión igual o superior a Ubuntu 11.04 no se deberá descargar ninguno de estos programas. En el caso de no estar ejecutando desde esta plataforma se pueden obtener ambos programas con los siguientes comandos:

```
$    sudo apt-get install libreoffic
$    sudo apt-get install gedit
```

Ejecutar el programa desde la interfaz gráfica:

- Para ejecutar el programa desde la interfaz grafica se han de realizar los siguientes pasos:
  - Pulsar el botón "*Player*" y luego el botón "*Simulación*". Si se ha realizado algún cambio en el código o es la primeara vez que lo ejecuta se deberá pulsar el botón "*Player-Make*" en lugar del botón "*Player*".
  - Si se desea utilizar la utilidad de la cámara se ha de pulsar el botón cámara en lugar del botón "Simulación".

Anexo II – Gestión del proyecto

Gestión temporal del proyecto

Durante las siguientes páginas se mostrará una planificación temporal y económica del proyecto que se ha realizado.

Una planificación temporal ayuda a conocer el estado del proyecto en cada momento (si la se mantiene una planificación medida y equilibrada). Si bien es muy difícil seguir una planificación exhaustiva del proyecto, es necesario mantenerla actualizada y ceñirse lo más posible a los tiempos establecidos. En función de las horas establecidas se generará un presupuesto de recursos humanos que deberá ser fiel a la planificación.

A continuación se muestra una tabla con las tareas realizadas.

Fases	Nombre de tarea	Duración
Estudio de viabilidad		**15d**
	Estudio de proyectos anteriores	5d
	Estudio de *Player/Stage*	5d
	Estudio de controladores	5d
Desarrollo del proyecto		**108d**
	Planificación previa del proyecto	1d
	Análisis de necesidades	12d
	Diseño del sistema simulado	20d
	Implementación del sistema simulado	30d
	Pruebas con el sistema simulado	10d
	Estudio del dispositivo real y las librerías OpenCV	5d
	Diseño del sistema con disp. real.	10d
	Implementación del sistema con disp. real.	15d
	Pruebas del sistema con el disp. real	5d
Post Mortem		**10d**
	Desarrollo del documento final + post mortem	10d

Las tareas del estudio de viabilidad son tareas realizadas por dos personas al mismo tiempo. Si bien el tiempo refleja un total de 15 días de realización, este tiempo fue estimado acorde a los recursos humanos de los que se disponía. En el caso de haberse realizado con una persona el total de tiempo para esta fase habría sido 30 días.

Durante el desarrollo del proyecto, las fases de planificación, análisis de necesidades y diseño del sistema simulado fueron realizadas también junto a Carlos Martín Martínez.

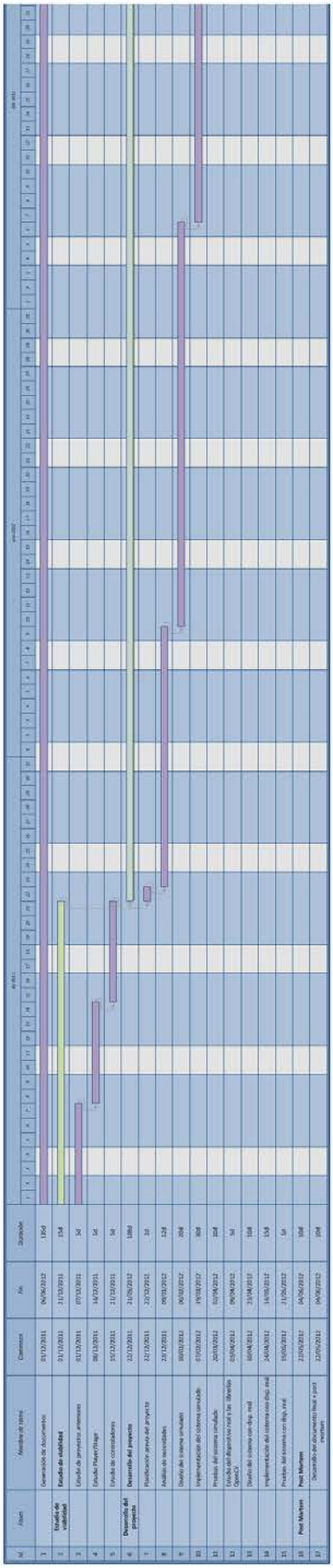
Durante la fase de pruebas de los sistemas se incluye también la planificación de las mismas.

La etapa de Post Mortem del proyecto engloba las tareas de reubicación de la documentación generada para realizar el documento actual, cambios en el sistema y la redacción del manual de instrucciones.

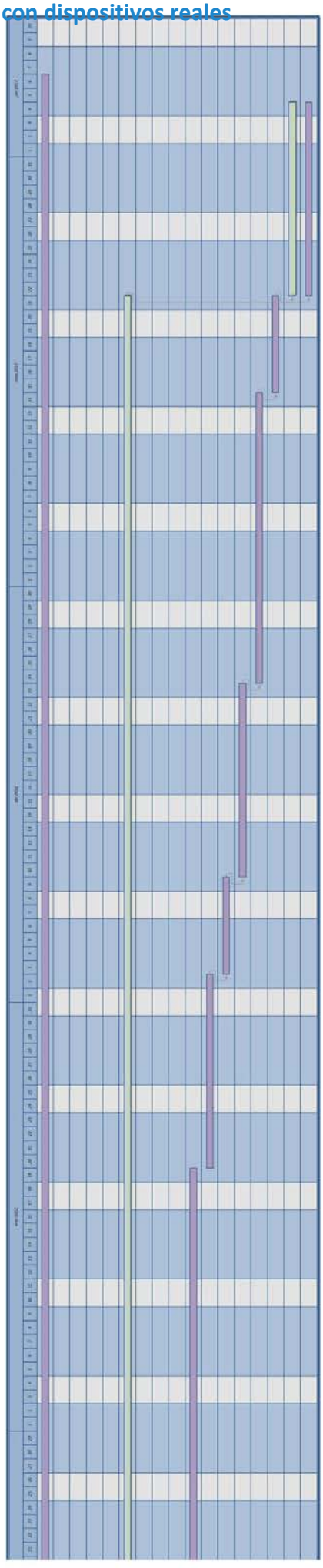
En la figura 30 se mostrará un diagrama de Gantt con las planificaciones.

A continuación se presentará una tabla con el presupuesto necesario en cuestión de recursos humanos. Se ha dividido en horas por ingeniero y estimado un precio basándose en el actual precio de consultor por hora en *Accenture* y el precio de un común como autónomo. El precio estipulado es de 30€ la hora

	Días	Horas	Precio
Ingeniero 1	48	48 x 6h/día = 288 horas	288 horas * 30 €= 8640 €
Ingeniero 2	181	181 + 6h/día = 1086 horas	1086 horas * 30 €= 32580 €
Total			**41220 €**



Id	Fase	Nombre de tarea	Comienzo	Fin	Duración
1		Generación de documentos	03/10/2011	04/06/2012	185d
2	Estudio de viabilidad	Estudio de viabilidad	31/10/2011	31/10/2011	15d
3		Estudio de proyectos anteriores	01/11/2011	07/11/2011	5d
4		Estudio Player/Stage	08/11/2011	14/11/2011	5d
5		Estudio de controladores	15/11/2011	21/11/2011	5d
6	Desarrollo del proyecto	Desarrollo del proyecto	22/11/2011	21/06/2012	180d
7		Planificación inicial del proyecto	22/11/2011	22/12/2011	1d
8		Análisis de necesidades	23/12/2011	09/01/2012	12d
9		Diseño del sistema simulado	10/01/2012	06/02/2012	20d
10		Implementación del sistema simulado	07/02/2012	19/02/2012	10d
11		Pruebas del sistema simulado	20/02/2012	02/04/2012	31d
12		Estudio del dispositivo real y las librerías OpenCV	03/04/2012	09/04/2012	5d
13		Diseño del sistema con disp. real	10/04/2012	23/04/2012	10d
14		Implementación del sistema con disp. real	24/04/2012	14/05/2012	15d
15		Pruebas del sistema con disp. real	15/05/2012	21/06/2012	5d
16	Post Mortem	Post Mortem	22/05/2012	04/06/2012	10d
17		Documentación del proyecto final y post mortem	22/05/2012	04/06/2012	10d



Gestión del hardware/software necesario para desarrollar el sistema.

Para poder desarrollar el sistema se necesita una cantidad muy reducida de recursos *hardware* que son:

Una ordenador con los siguientes requisitos mínimos:

1. Un procesador de un solo núcleo (se recomienda utilizar varios núcleos para futuras implementaciones).
2. Memoria RAM 2GB
3. Cámara web.
4. Disco duro de 100GB.
5. Tarjeta gráfica de 64 MB de memoria.

En cuanto a la gestión software del sistema necesita:

1. Sistema operativo Ubuntu10.01
2. Librerías OpenCV 2.1
3. Herramientas *Player/Stage*
4. Microsoft Word, Visio, PowerPoint. (paquete Office 2010).

A continuación se muestra una tabla presupuestaria para la gestión de software/hardware total. El hardware necesario se puede obtener a partir de ordenador portátil básico así es posible abaratar costes. Los precios han sido obtenidos de la página http://www.dell.es y http://office.microsoft.com .

Tipo	Producto	Precio
Hardware		**499 €**
	Ordenador Portatil Inspiron 15'' Dell	499 €
Software		**139 €**
	Sistema operativo Ubuntu 10.01	0€
	Librerías OpenCV 2.1	0€
	Herramientas *Player/Stage*	0€
	Paquete Office	139 €
Total		**638 €**

Gestión económica total del proyecto:

Recursos	Precio
Recursos humanos	41220 €
Recursos hardware/software	638 €
Total	**41858 €**